A Responsive Schedule Management System for Large Homecare Organizations

Chakar, T.M. (1397818)

taner.sezgin@gmail.com

Program: Master Engineering and Policy Analysis, Faculty of TPM

Delft University of Technology (TU Delft)

Supervised by:

- Alexander Verbraeck (TU Delft)
- Joseph Barjis (TU Delft)
- Scott Cunningham (TU Delft)
- Ali Tamer Unal (BOUN)

Rev: Tuesday, 07 March 2011

Table of Contents

SUMMARY **5**

1. INTRODUCTION **5**

2. PROBLEM EXPLORATION **7**

2.1. Difficulties in scheduling for LHO 7
2.2. How schedule software can help to overcome difficulties? 8
2.3. The software requirement of LHO 10
2.4. The problem 10

3. RESEARCH **13**

3.1. Problem statement 13
3.2. Research objective 13
3.3. Research questions 13
3.4. Solution approach 14
3.5. Research Methods and Tools 17
3.6. Structure of the report 18

4. SCHEDULING SOFTWARE SCOPE **20**

4.1. The environment 20
4.1.1. Basic Processes and Terms 20
4.1.2. The organization 23
4.2. The software requirement of LHO 23
4.3. The scope 25

5. LITERATURE REVIEW **26**

6. OBJECT MODEL **30**

6.1. Scheduling system separation 30
6.2. Relevant Ontologies in the literature 31
6.3. Class Diagram 31
6.3.1. Organizational Classes 31
6.3.2. Scheduling Classes 34
6.3.3. Rule Support Classes 35
6.3.4. Shared resource classes 36
6.3.5. Distributed Decision Making Support Classes 36
6.4. The distributed decision making and authorization process 37

7. SCHEDULE CONSISTENCY AND PROCESS MANAGEMENT **38**

7.1. Importance of schedule (data) consistency 38
7.2. Alternative data protection methods 38
7.3. Why centralized model for LHO? 40
7.4. System Configuration 41
7.5. Implementation of data locking to LHO 41
7.5.1. Data locking mechanism 42
7.5.2. The Data Locking unit (atomic unit) 43
7.5.3. Implementation of data locks 46
7.5.4. Object Locking 47
7.6. Software task prioritization 48

7.7. SOFTWARE TASK ABORTS 48
8. SOLUTION IMPLEMENTATION 50
9. CONCLUSION 55
10. REFLECTION 57
11. BIBLIOGRAPHY 59
12. APPENDIX 1: USE CASES 63
13. APPENDIX 2: STATISTICS AND COUNTERS 72
14. APPENDIX 3: CONSTRAINTS 73
AN EXAMPLE SCHEDULE 73
Schedule from the employee perspective 74
Schedule from the client perspective 75
Schedule from the organizational/administration perspective 76
15. APPENDIX 4: TEST RESULTS 77
RPDS SCENARIO 77
BDS SCENARIO 78
GLS SCENARIO 82

Figures

Figure 1: Example GANTT chart for Area 1 12
Figure 2: The Rational Unified Process 16
Figure 3: Structure of the report 19
Figure 4: Basic process definition for LHO 20
Figure 5: Simplified organizational structure 23
Figure 6: Employee related classes 32
Figure 7: Client related classes 33
Figure 8: Organizational structure classes 33
Figure 9: Scheduling classes 34
Figure 10: Rule support classes 35
Figure 11: Classes related to shared resources 36
Figure 12: Distributed Decision Making Support Classes 36
Figure 13: Client-Server configuration 41
Figure 14: Lock acquisition procedure 42
Figure 15: Lock prioritization 43
Figure 16: Thread states (with and without locks) 45
Figure 17: Thread states with global lock 46
Figure 18: Average response times for RPDS 52
Figure 19: User ratings for RPDS 52
Figure 20: Average response times for BDS 53
Figure 21: User ratings for BDS 53
Figure 22: Average response times for GLS 54
Figure 23: User ratings for GLS 54
Figure 24 GANTT chart for Taner Chakar 75
Figure 25: GANTT chart for the Client 1, Client2 and Client 3 76

Tables

Table 1: SWOT analysis for scheduling system implementation 9
Table 2: Methodology summary 17
Table 3: Homecare terms 21
Table 4: Categorization of the available literature 29
Table 5: Thread states 43
Table 6: Test results 51
Table 7: Example schedule for one employee Taner Chakar 73

Summary

The planning and scheduling process of large homecare organizations (LHO) must be responsive as LHO cannot afford postponement and cancellation of services. However, achieving such responsiveness (respond to schedule change requests without violating timing constraints) is a very complex task for such large organizations, due to their extensive geographical spread, fuzzy objectives and uncertain environment. In our research we developed a software architecture which will help LHO to achieve responsive scheduling process. We defined the architecture in four steps. As the first step we proposed to separate the schedule into independent sub schedules. As second step, we proposed scheduling system object model which is suitable for handling of low-level scheduling processes in distributed, multi-user environment. Thirdly, we proposed a centralized scheduling system which can handle shared resources and high level organizational policies. In order to protect the consistency of the centralized schedule, we proposed to use data locks. We propose to use Area (or Team) as atomic locking unit in order to ease the maintenance of the software and to have responsive scheduling process. As fourth and last step of the architecture, we propose to break long taking software tasks into subparts so that the other software tasks can also be processed without waiting too long in the queue.

Keywords: homecare, responsive, scheduling, distributed, multi-user

1. Introduction

Homecare is the umbrella term for care services that take place at a patient's home. These care services include a wide range of supportive services, ranging from help with basic daily activities (e.g. help with bathing, eating, walking or going to the toilet) to medical activities like nursing.

In the last decade, there is an increasing demand for homecare in Europe, as the number of elderly people has grown continuously. According to EU statistics, the number of elderly people will continue growing for the next 20 years, thus increasing the demand for homecare (see 'Eurostat regional yearbook 2009' for details).

Homecare services were originally introduced with a need to improve responsiveness, continuity, efficiency and equity (Baris, 2008). With the introduction of homecare services, demand for hospitals declined, thus decreasing the domestic healthcare costs spend on elderly care. However, with the growth in demand, the homecare costs rose dramatically. The homecare costs currently comprise a major part of the total healthcare costs. With the intention to reduce national budget deficits, several European governments are searching for appropriate austerity measures, some of which are to be found in their healthcare programs.

As the reduction in healthcare budgets will likely result in lower profit margins for homecare organizations, it is very important for homecare organizations to be cost effective. One of the ways to reduce the costs within homecare organizations is to increase the efficiency of the core business. As the scheduling of activities is intimately related to the efficiency with which core business activities are performed, Scheduling has become a very important element for homecare organizations and a strong potential tool for cost reduction.

Unfortunately, the scheduling of large homecare organizations is not such an easy task. First of all, the problem size is very big due to the amount of employees and patients active at any one time. Secondly, the scheduling is related to humans, subjects that are notorious for their fuzzy objectives and soft constraints. Finally, the scheduler or the scheduling system must be responsive (respond to schedule change requests without violating timing

constraints) to urgent tasks because the homecare environment is very uncertain and requested services cannot be postponed or cancelled.

Achieving such responsiveness is a very complex task for Large Homecare Organizations (LHO, organizations with more than 100 employees which operates in more than one location is considered to be large) since they have to deal with distributed, co-dependent decision making. LHO are distributed in wide geographical areas and so are the decision makers (planners). But decisions that one of the planner makes, influences the other.

In this Thesis, we will develop an architecture for a LHO scheduling software system which will help LHO to achieve responsive schedules. The proposed model will define an object model for a scheduling system and propose a systematic method to manage low-level processes in a distributed, multi-user environment. Afterwards, the proposed model will be tested with a pilot organization, and results will be examined.

In the literature, there are solutions which may be used to solve some parts of the problem that we study but there are none that accommodate all of the problem characteristics here present, rendering them obsolete for direct and integral application to the LHO problem (see section '5. Literature Review' for details).

The rest of the thesis is organized as follows: the next section of this document will frame the problem and define its context. It will be followed by an explanation of the Objective of the research and research questions. The fourth section will define the scheduling software scope. The fifth section will present the literature findings. The following four chapters will define the solution model and illustrative implementation. The last chapter is dedicated to conclusions.

2. Problem Exploration

This chapter will define the architectural requirements of the LHO scheduling software system and it will address the problems that arise around it. In order to do so, we first define the LHO environment. Next we will describe the scheduling problems in LHO from different perspectives and define the scheduling system requirements. The last section of this chapter will define the research problem of this study.

2.1.Difficulties in scheduling for LHO

In this study, we define the scheduling process as the process of matching the service requirements (demand) with company resources. The demand is expressed in the form of tasks. Employees are the company resources which perform the tasks. The schedule of the LHO is obtained by assigning the tasks to the employees. The scheduling process consists of 2-10 weeks. This type of scheduling is generalized as operational planning and dispatching.

A **good plan** is defined as one that is capable of collectively matching the expectations of the clients, employees and the organization. However, the matching of expectations is a difficult process because the size of the scheduling problem is very big and it involves many different human actors who have fuzzy objectives. We summarize the scheduling difficulties of LHO into five main items:

1. The size of the problem is very big

On average, each employee visits 2 clients per day. In order to create a plan for one week, the planning department of LHO with 10000 employees must plan 100000 tasks. Such a workload is far too demanding to be handled centrally by one person. Alternatively, the scheduling problem can be solved using optimization techniques or heuristics. But the solution time may take hours, even days. As the scheduling system of LHO is expected to be responsive, implementing such techniques alone will not be a solution to the problem. For instance, imagine that an employee calls the planning department and reports that she is sick. After getting this input, the planner needs to replace her with another employee within few minutes because the next client might be in urge. For example, there are clients who cannot feed themselves or cannot go to bathroom on their own. Their services can't be skipped or postponed.

2. There are many constraints

The basic job of a LHO planner is to match expectations of the actors involved. In the LHO case, the most crucial actors for planning are employees, clients and the LHO organization. Each of these actors imposes different constraints on the planning process. For instance, an employee expects that she doesn't work more than her contracted hours; that her preferences are considered; she gets enough rest, etc. A client expects that her requests are satisfied on time. A LHO organization expects to satisfy all clients on time. It is possible to extend the list of constraints. During our analysis, we came across more than 15 major constraints. These constraints are demonstrated with an example in "Appendix 3: Constraints".

From the list of constraints provided in "Appendix 3: Constraints" we can conclude that the number of constraints which need to be considered when planning a task is far greater than human mind can handle.

3. The scheduling process must be responsive to urgent requests

Especially the short term planner (in the dispatching phase) must respond to state changes within a minute or two. For instance, sometimes employees can't find the client at home because he is taken to hospital. In such a situation, the employee calls the dispatching centre and asks for instructions. The planner needs to analyse the situation and assign a new task to the employee in a few minutes. Otherwise, employee capacity will be wasted.

4. The objectives are fuzzy and the constraints are soft

The objectives of the employees or clients cannot be defined concretely as they are humans. The human mind may behave very differently under different circumstances. It is well known fact that an employee may have different objectives over time. For example, an employee may not prefer to do overtime. However, she may consider doing overtime if she needs extra money.

In addition, the human mind is different from that of a machine. The human has a lot more states than "working" or "non-working". This implies that employees may have soft constraints which are case dependent and cannot be generalized. For instance, quite commonly, people are specified as morning person or evening person. A morning person is more efficient if she starts working early. Similarly, evening person will prefer to start working late. The planner must consider such states of mind when making a plan.

5. There are multiple planners who work at distributed locations.

The planning of LHO cannot be handled by a single planner because the problem is too big. Therefore the schedule of LHO is created by several planners. Each planner is responsible from one part of the planning. The most common way is to assign a planner is on a geographical basis (e.g. to each **Area)**. An Area is defined as "*a team of employees who have a similar set of skills and deliver services to the clients located in the same geographical area*". However, the planning of Areas is not independent as employees/clients can have multiple contracts with different areas. Therefore the planners of different Areas need to synchronize their planning in order to avoid conflicts like over-planning, etc.

2.2.How schedule software can help to overcome difficulties?

The Scheduling software has various benefits for the scheduling process which may help to overcome the difficulties described in the previous section. Moreover, it may provide improvements on many other aspects. Table 1 provides SWOT analysis about implementation of scheduling software at a homecare organization.

Table 1: SWOT analysis for scheduling system implementation

Strengths	Weaknesses
• Easier and faster schedule alterations • Timely and accurate schedule outputs • Cost reduction via increasing efficiency and decreasing administrative costs • Possibility of implementing complex solutions like optimization, heuristics, etc.	• High cost (software license, implementation costs, maintenance fees) • Result of the outcome depends on data inputs • Requires commitment from many parties, not only planning departments
Opportunities	**Threats**
• It will steer improvements in data collection process, which will result in many benefits like timely and accurate invoices.	• Planners and other employees can be threatened by losing their jobs • Employee unions may oppose to various processes like data collection

Although, the benefits of the organization will vary based on the industry and size of the company, it is possible to generalize the benefits of the scheduling software for the planner (or planning department). We list five major points:

- The Scheduling system can help to retrieve accurate and timely inputs by interfacing to different data sources like backbone systems, payroll systems, spread sheets, personal PDA's, etc. This will reduce enormous effort from planner's shoulders.
- The software may generate automated schedules using various techniques like optimization, heuristics, etc. These techniques may generate schedules for the organization without any human intervention.
- The scheduling system may provide an opportunity for the planners to manually/semi manually generate schedule in electronic environment. In most of the cases, automated schedule generation techniques are not sufficient and comprehensive enough to cover all the objectives and constraints, e.g. fuzzy objectives or soft constraints. In such cases, a planner needs to interfere with the schedule and manipulate it manually. Or in dispatching process, the planner may only require only manual scheduling support if she wants to have control on the schedule at all times.
- The software may provide decision support by signalling constraint violations, calculating aggregate counters, keeping track of unutilized resources, etc. Such calculations can be performed very efficiently by a software and make the planner's life very easy.
- The software may help with visualization of schedule outputs. It may display the same schedule in various forms. There are many stakeholders who want to see the planning in their own customized way. For instance, an employee will want to see only her tasks, a client wants to see when she receives help, a customer relations department will want to extract automatically created letters in order to inform clients about the planning, etc.

In our study, we will look for a software architecture which is suitable for scheduling systems which provide the benefits mentioned above. Specifically, we will concentrate on large home care organizations. In LHO, the decision makers (planners) are working at distributed locations but they control the shared resources.

In the next section, we will specify briefly the homecare scheduling software requirements. Then we will explain the problems around the LHO scheduling software architecture.

2.3.The software requirement of LHO

LHO usually deploys scheduling software to solve its complex scheduling problem. The scheduling software intends to support decision makers by providing different sorts of information, generating schedules, calculating scheduling outputs, etc.

In this section, we will summarize the most important scheduling software requirements of LHO. The extended and complete list of the requirements can be found in section *4.2 The software requirement of LHO*. We will consider operational planning and dispatching (see section *4.3 The scope* for details) and we will concentrate only on low-level scheduling processes. When defining the scheduling requirements, we will exclude data interfacing requirements.

The LHO scheduling system requirements can be described in three major items:

- **Functional requirements:** LHO scheduling software is expected to support low-level scheduling processes, e.g. plan a task, plan a holiday, etc. Each use case in "Appendix 1: Use cases" represents a low level scheduling process. Each use case usually inherits a set of rules/constraints. The scheduling system must support the LHO rules, constraints and objectives. The rule/constraint handling requires the handling of information on various levels of aggregations.
- **Non-Functional requirements:** The software is expected to support a multi-user architecture because there can be multiple planners who share responsibilities. The planners are located at different offices. Therefore the software is expected to support distributed decision making. Most importantly, the software must be responsive in order to support fast decision making. We define responsive scheduling system as a scheduling system which responds to schedule change requests without violating the timing constraints. Each use case of the scheduling system will have a timing constraint which defines the expected runtime of the use case.

2.4.The problem

Based on the software requirements given in the previous chapter, we can conclude that LHO need a responsive scheduling system which will manage low-level processes in a distributed, multi-user environment. In our study, we will look for architecture for LHO scheduling software system which will help LHO to achieve responsive schedules. We define an architecture in two parts: (i) an object model and (ii) a systematic method to manage processes (software tasks) in a distributed, multi-user environment.

The first challenge is to find the suitable object model for the LHO scheduling software. The scheduling problem of LHO needs to be decomposed as it cannot be solved as one piece due to difficulties we mentioned in section "2.1 Difficulties in scheduling for LHO". Therefore, a suitable object model depends strongly on the decomposition of the whole problem. The decomposition level depends on a large number of factors like decision makers, their

responsibilities, the aggregation level of decision making, etc. In other words, the decomposition strongly depends on the LHO environment.

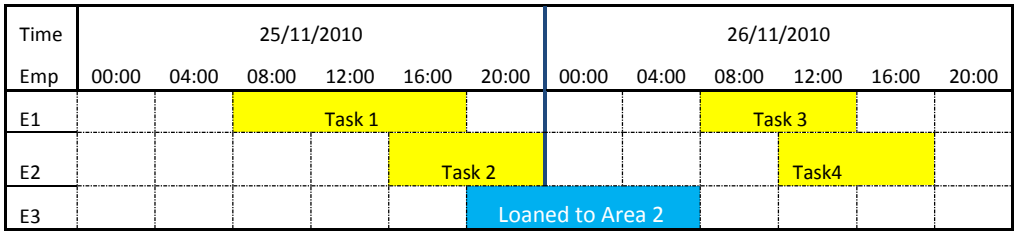
The LHO scheduling software cannot be decomposed into very small (atomic), independent pieces which concern only one decision maker because different levels of aggregation are required for decision making. Planners are responsible for making global and/or local decisions. For instance, when planning the summer holidays of the employees, the planner must control not only the available capacity of her region but also the available capacity of the entire city. Otherwise the capacity in the summer time may drop below the demand. Or when there are shared resources among Areas, e.g. employees who are working for different Areas, then a decision of one planner will influence the decision of another.

However, the atomic unit cannot be very large because it will reduce the responsiveness. In order to protect the feasibility of the schedule, the software will try to process the software requests sequentially instead of parallel. Consequently, the responsiveness of the software will decrease. Imagine there are N planners in the organization who want to work simultaneously. Then the software will process the request of one of the planner while the rest (N-1) planners will wait. Or one of the planners will need to wait for N-1 planners. In Europe there are organizations with more than 400 planners. It means that one of the planners will have to wait until the tasks of the previous 399 planners are executed by the software.

Therefore, the way that the software tasks will be processed is very important for the feasibility of the schedule. Moreover, it has tremendous influence on the responsiveness of the scheduling system. However, defining a systematic approach to prioritize software tasks is difficult due to resource sharing requirements and different scheduling rules on different levels of aggregation. For instance, registering one employee as sick seems to affect only the employee herself. However, it will also affect the other *Areas* where the employee is working. In addition, it will also influence the other neighbouring *Areas* since it will affect the total available capacity of the region. And if there are tasks which are at the time of the sickness, then the clients of these tasks will also be affected.

We will illustrate the problems described above with a simple example. Let us assume that the GANTT chart given in Figure 1 represents a schedule for Area 1. In a typical LHO, this plan may be created by several planners. For our example, let us assume that a short term planner (STP1) is responsible for generating a schedule for 25/11/2010 and long term planner (LTP1) is responsible for generating a schedule for 26/11/2010. If STP1 makes a change on the schedule of the E1, then the LTP1 has to be notified about the change. Otherwise they may make conflicting decisions. For instance, assume that STP changes the end time of the task on E1 on 25/11/2010 to 24:00. The result will be that E1 will not have enough rest as she starts to work at 08:00 in the next day. Therefore the LTP needs to be notified that she needs to change the schedule of E1. The situation is even more complicated for E3. E3 is loaned to Area 2 between 25/11/2010 20:00 and 26/11/2010 04:00. Assuming that STP2 has control over Area 2 on 25/11/2010 and LTP2 has control over Area 2 on 26/11/2010, we can say that any decision concerning E3 on 25/11/2010 and 26/11/2010 will influence 4 decision makers. In practice, there are LHO organizations with over 1000 Areas. This creates a huge network of decision makers whose decisions are not totally independent. Moreover, these decision makers are working in different locations, i.e. physical communication is not easy.

Figure 1: Example GANTT chart for Area 1*



*Each row represents one employee. Yellow boxes demonstrate tasks. Blue box represents a time where employee works for different Area.

In this study, we are going to define a scheduling software architecture which will help planners avoid conflicting decisions and still be responsive. As our example demonstrates, it is not possible to separate schedules of different Areas. However, not decomposing the problem will also cause responsiveness issues. Therefore, we seek for an object model which is suitable for a LHO environment and which will support distributed decision making with multiple users.

We can summarize the problem as:

"LHO requires a responsive scheduling software system to support its low-level scheduling processes. Achieving responsiveness for such scheduling software is a very complex task since LHO environment includes distributed, co-dependent decision making. The LHO scheduling software must be able to operate in a distributed, multi-user environment and must comply with timing constraints (expected runtime of a use case)."

In the literature, there are solutions which may be used to solve some parts of the problem but they can't be applied directly to LHO problem. Itabashi et al. (2005), Corchado et al. (2008), Fraile et al. (2009), Huang et al. (1995) and Date and Matsuo (2008) explored the homecare scheduling area using distributed and/or multi-agents systems. However their models are not directly applicable to our problem (See '5. Literature Review' for details).

A number of papers in the homecare literature concentrate on optimal scheduling of the homecare organizations. This approach is helpful for our problem since it may reduce the need for changing the plan by improving the initial plan. However, it will never remove the requirement for real time system which handles daily schedule changes that requires distributed, multi-user handling. The models that are most relevant to our study are Eveborn et al. (2006), Borsani et al. (2006), Hutzschenreuter et al. (2008) and Begur et al. (1997). In the Literature Review chapter we discuss in detail the overlap of the existing literature with our problem.

3. Research

This chapter will define the research questions and an approach to solve these questions. First, we will define the scope of the research. Next, the problem statement, the research objectives and the research questions will be formulated. At the end of the chapter, we will explain our solution approach to the research problem.

3.1. Problem statement

As a summary of the knowledge gaps provided in previous sections, the problem statement of this research can be defined as:

"LHO needs a responsive scheduling software system which will support the management of low-level processes in a distributed, multi-user environment."

Key concepts:

Responsiveness: Real-time systems are expected to complete software tasks within specified time limits. The time limits are called timing constraints. In LHO case, there are soft timing constraints (Santos et al., 2008) for each use case. For example, the planning of one task should take no longer than 2 seconds. Soft timing constraints can be violated. However, the violation of the soft constraints may cause significant damages. For instance, a planner needs to respond within a few seconds to an employee who is waiting on the phone for a new schedule. Otherwise, the organization will underutilize its resources. We define a responsive scheduling system as a system which performs without violating timing constraints.

Consistency: Consistent schedule is one that does not contain a contradiction. Contradiction occurs when: (i) two or more planners make decisions which result in an infeasible schedule (ii) when planner makes a decision based on obsolete information.

For example, imagine that a short term planner (STP) registers an employee as absent for the next 15 days. Infeasibility will occur if the long term planner (LTP) plans the employee for the same period.

In order to avoid contradiction, both planners should see the same schedule. In other words, a planner must be notified if other planners change the planning of her Area.

3.2. Research objective

The objective of this thesis will be:

"To provide LHO with an architecture for a scheduling software system that will achieve responsive and consistent schedules. A scheduling system must support management of low-level processes in a distributed, multi-user environment."

3.3. Research questions

This thesis will seek an answer of the following question:

"What is a suitable software architecture for large and dynamic homecare organizations to achieve a responsive and consistent scheduling process in a distributed, multi-user environment?"

Sub questions:

1. Can we separate the schedule of different locations into independent sub schedules?
2. What is a suitable object model which will support scheduling processes in a distributed, multi-user decision making environment?
3. How can the scheduling system handle the scheduling rules and constraints on different aggregation levels in a distributed, multi-user environment?
4. How can we ensure the consistency of the schedule in a distributed, multi-user environment without worsening responsiveness?
5. How can we prioritize LHO scheduling software tasks in a distributed, multi-user environment in order to achieve responsiveness?

We define the architecture through an object model and a mechanism to support low-level process management in a distributed, multi-user environment. In order to solve our problem, we first seek a possible separation of schedule. After this we seek for a suitable object model for the independent sub schedules. Then we research the way to manage the software tasks. First we need to find a way to structure/decompose the scheduling system. Then we need to define concurrent access procedures. These procedures are important because they will have big influence on responsiveness. Then we need to define a way to prioritize software tasks when there are software tasks with different execution times.

3.4.Solution approach

The software quality problems are a 100 to 1000 times more expensive to find and repair after deployment than before Kruchten (2004). We expect that using traditional linear methods like 'the Waterfall method' for our research will result in high project costs as they push the risks forward in time. The cost of undoing mistakes later in the project is lot more costly than undoing them early in the process. Instead we focus on iterative approach which will allow continuous verification of the software quality and reduces the risk of costs incurring late in the project phase. An iterative process is required for our research because the research requires the matching of conflicting goals of different actors. For example, management expects cost reductions while employees expect convenient and flexible working times. In addition, iterative approach will be beneficial to define the software requirements as there is no single language which can be used to define technical requirements and functional requirements. Therefore misunderstandings and miscommunications can occur during the analysis process which can be corrected earlier in the project by using an iterative approach.

As a solution approach, we will follow The Rational Unified Process (RUP) which is an iterative model that is created based on similarities between different software development projects across the world has led to the construction of a so-called Rational Unified Process (RUP) framework that aims to capture the temporal character of the different tasks to be performed (see Figure 2). By setting out the different tasks (vertical axis) as a function of time (horizontal axis), common task overlaps become visible due to the iterative development that is commonly observed between the tasks. Through this conceptualization, it is possible to identify different phases along the timeline that are common to most software development projects.

The inception phase is primarily characterized by an iterative process of both business modelling and the formulation of software requirements. Once the initial requirements are

roughly formulated, an elaboration phase sets in. In this phase, the business modelling and requirement formulation are elaborated while an initial analysis and design, implementation and testing is conducted. It is even possible to commence the initial steps for deployment at this stage. Through an iterative process of mutual adjustment and development, software construction itself takes off. At this stage, the implementation of software takes the overhand, translating the software requirements into code that is to be tested and eventually deployed. Through an iterative process of analysis and design, implementation and testing, the deployment of the software takes the lead in the last phase, the transition.

As the Figure 2 figure shows, the RUP process dictates that software development is performed in six tasks: (i) Business modelling, (ii) Requirements, (iii) Analysis and design, (iv) Implementation, (v) Test and (vi)Deployment. Figure 2 provides an overview of RUP steps over time.

The first two tasks of the RUP, Business modelling and Requirements, are the motivations for this study. Brief description of the Business modelling and Requirements steps is provided in the first two chapters of the thesis.

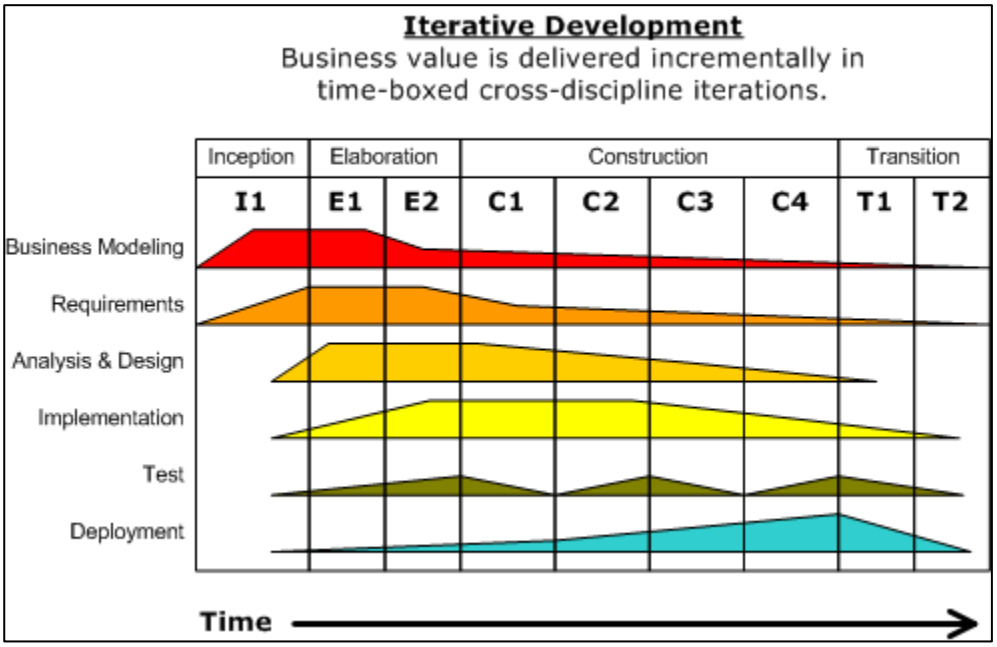
For the analysis task, we will follow an iterative software development process because the analysis process of the architecture is very complicated. The analysis process is complicated because it requires involvement of many parties with different expectations and goals. For example, planners expect the software to help them in the planning process, while management expects cost reductions. In addition, software requirements need to be defined in a more abstract and technical level while the analysis language is usually more functional. Therefore misunderstandings and miscommunications can occur during the analysis process.

As a result of analysis process we define the scheduling software requirements. We perform the analysis process with two LHO (both employ over ten thousand employees) located in North Europe. We did a series of interviews with various stakeholders including the management, planners, finance managers, etc. Most of the time, it is hard to gather all the stakeholders in one meeting. Therefore we also had to scan many internal documents on the scheduling system definition.

The design process is also complicated because the impacts of the designs are not predictable. For example, the impact of various atomic locking units is not easily predictable as there is a trade-off between using low level locks and spending more time for the acquisition of locks, using high level locks and waiting for locks.

As first step of the design process, a literature review (or desk) research is performed to gain insight into the problem. Then we search the answers of each sub-question in the literature. After the literature research, we continue a design process that aims to design an object model and construct a mechanism for the management of software tasks. We perform the design phase in three steps. First, we design the complete architecture in a less detailed manner to oversee the integration problems from the beginning. Second, we design individual components of the research and share it with other developers and planners. By taking this step, we try to eliminate functional misunderstandings and possible bugs. Third, we change the design (if necessary) according to the feedback. We will call this method **Iterative Architecture Development.**

Figure 2: The Rational Unified Process*



*Source: Downloaded from http://en.wikipedia.org/wiki/IBM_Rational_Unified_Process#Six_Engineering_Disciplines on 20th of January, 2010

We outsource the implementation phase, where the designed architecture is implemented as software.

In the next step, we test and validate the architecture that we propose with the implementation done at Company A.

Table 2: Methodology summary provides summary of methodologies which are going to be utilized to answer each question.

Table 2: Methodology summary

Question	Methods
How can we separate the schedule of different locations into independent sub schedules?	1. Desk research 2. Iterative Architecture Development
What is a suitable object model which will support scheduling processes in a distributed, multi-user decision making environment?	1. Desk research 2. Iterative Architecture Development
How can the scheduling system handle the scheduling rules and constraints on different aggregation levels in a distributed, multi-user environment?	1. Desk research 2. Iterative Architecture Development
How can we ensure the consistency of the schedule in a distributed, multi-user environment without worsening responsiveness?	1. Desk research 2. Iterative Architecture Development
How can we prioritize LHO scheduling software tasks in a distributed, multi-user environment in order to achieve responsiveness?	1. Desk research 2. Iterative Architecture Development

3.5.Research Methods and Tools

In order to answer the questions mentioned above the following methodology will be used:

Desk research

Literature survey will be done to investigate existing solutions on the problem. Furthermore, external limitations will also be researched from scientific papers, journals, governmental publications, etc. In addition, we scan various company documents in order to define software requirements.

Direct interviews and interviews with interviewer

We did a series of interviews with various stakeholders including the management, planners, finance managers, etc. Most of the time, it is hard to gather all the stakeholders in one meeting. Therefore, from time to time we do interviews with analysis group representatives or other analysts who interviewed the analysis groups beforehand.

UML diagrams

The UML technique will be used to present our software architecture proposal. The expected output will be a class diagram and use cases of proposed schedule management and sharing software. In addition, state charts will be used to illustrate the distributed decision making and prioritization of tasks.

ICRON:

Icron software will be used to develop the model and perform tests. It has its own language to model scheduling processes. More information about company and the product can be found at http://www.icrontech.com/.

3.6.Structure of the report

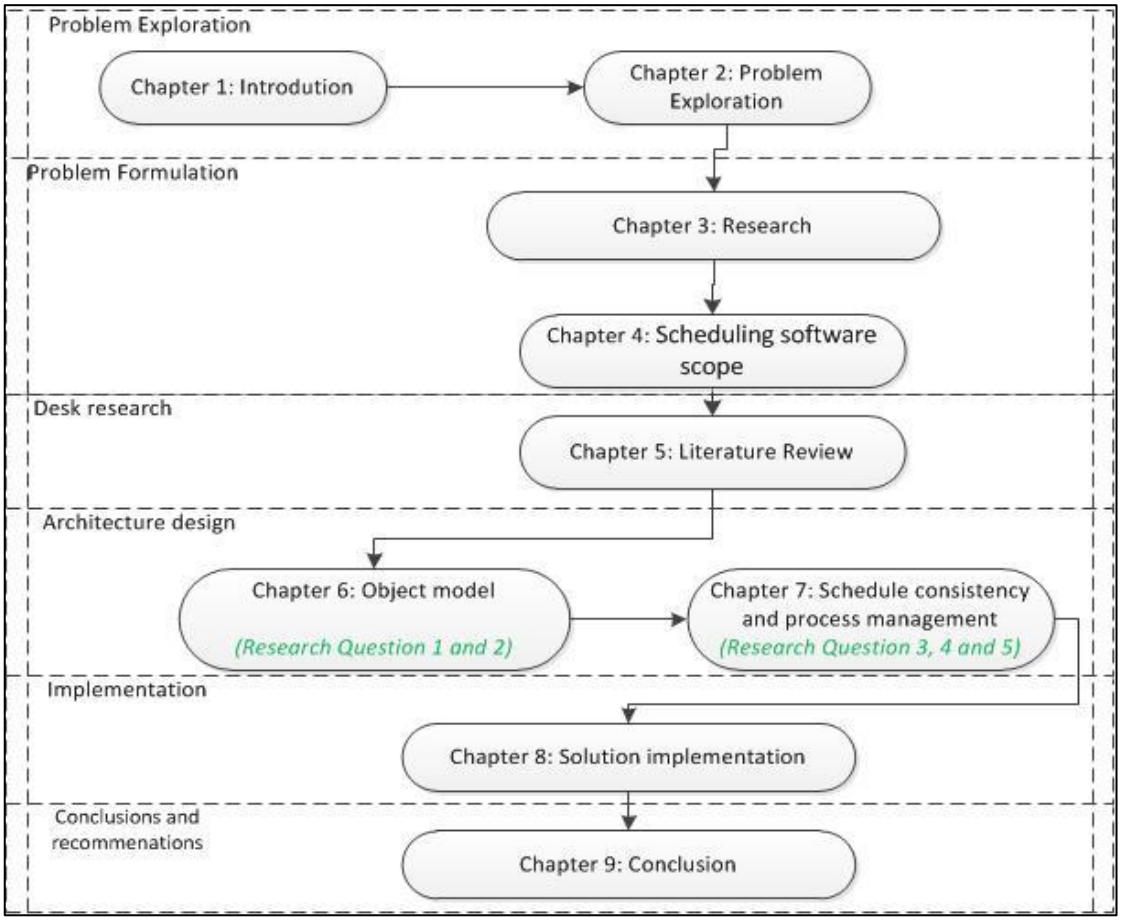
Figure 3 illustrates a map of the report. In the first two chapters, we explore the problem. In the third chapter, we formulate the problem statement and research questions. In the fourth chapter, we define the LHO software requirements in detail and indicate a scope for the study. In the fifth chapter, we will present the literature findings. In the following two chapters we will describe the results (the architecture) of the analytical research we conducted in order to find a suitable architecture for LHO scheduling software. We divide architecture into two parts: object model and process management.

In Chapter 6 we will define an object model for a scheduling system which will support a distributed, multi-user environment. The scheduling problem of LHO is too large to be managed as a single problem. Therefore we first look for a method to simplify the scheduling problem by separating the whole problem into independent sub problems. This way, we can define a separate scheduling system for each problem separation. After separating the problem, we will define the software system architecture of each separation.

In chapter 7 we will define the methods and procedures that will be deployed to ensure scheduling data consistency in a scheduling system. We will also advice a method to prioritize software tasks in a scheduling system when the processing time of the software tasks is variable, i.e. there are software tasks with short processing times and software tasks with long processing times.

The last two chapters are dedicated to the solution implementation and conclusion. In Chapter 8, we will describe the results of the pilot implementation. In Chapter 9 we will conclude the results of our findings.

Figure 3: Structure of the report



4. Scheduling software scope

This chapter will describe the software requirements of LHO and determine the scope of this study. First, we define the LHO environment. Next we will define the scheduling system requirements. The last section of this chapter will define the scope of this research.

4.1.The environment

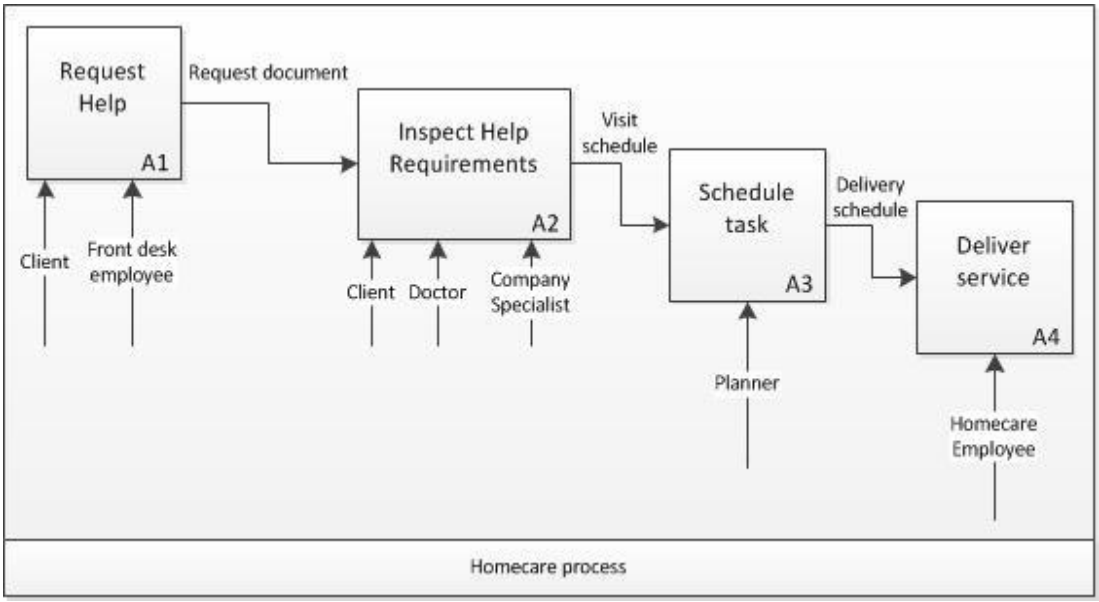
This section will define basic processes and terms that are required to describe the environment of LHO. It will be followed by the organizational structure of a LHO.

4.1.1. Basic Processes and Terms

Figure 4 displays the basic process steps from initiation of a help request until the help is delivered. The first step in a homecare service delivery is the Help Request. At this stage the *Client* requests help from the homecare organization. In the next step, a *Care Inspector* visits the *Client* and is accompanied by the client's doctor if necessary. During the inspection process, the *Care Inspector* and the *Client* decide on a *Visit Schedule* which defines the *Client Contract*. In addition, the *Care Inspector* assigns the visit request(s) to the most suitable *Team*(s) in the organization.

In the third step, the *Planner* of the *Team* assigns the *Task*(s) to the most suitable *Employee*(s). In the fourth step, the assigned *Employee(s)* execute(s) the *Task*. The third and fourth steps are repeated until the *Client* is recovered.

Figure 4: Basic process definition for LHO



For the sake of clarity, we give the definitions that are used in this Thesis in Table 3:

Table 3: Homecare terms

DEFINITIONS
Client A client of LHO is generally a physically injured person who cannot perform daily activities such as showering, cleaning, etc. without assistance of someone else. There are many types of services that are requested by the clients. However, this research will focus on the following service types: 1. Health care: The treatment at home. This also includes nursing services. 2. Personal care: Personal care can be summarized as helping the client to perform her daily activities. For instance, helping a client to take a shower in case of physical injuries, etc. 3. Domestic help: These are side help services which are delivered to clients who cannot perform the tasks themselves. For instance, old people or people with physical injuries who cannot cook for themselves or cannot do shopping, etc. 4. Psychological support: It is delivered to the clients who need psychological support because of a trauma, etc. 5. Babysitting: This is also a side service that is delivered in case that the parent is injured and cannot take care of the baby. 6. Cleaning help: Cleaning is also considered to be side service which is delivered to the clients who are not able to clean their homes themselves.
Client Contract Each client has a contract with the homecare organization which specifies what services are going to be delivered at what time frame.
Care Inspector A Senior employee who is responsible for the estimation of visits that is required for the client recovery. She is also responsible for negotiating visiting hours with the client.
Visit Schedule A List of tasks (task is defined below) that needs to be fulfilled. Each task has an earliest start time, a latest finish time and a defined duration. The task needs to be performed within the earliest start and latest finish time. The Task duration can be shortened in specific cases but these are considered out of scope for this study.
Task A task is defined as a help activity has to be performed within a certain time period at certain location. Every task can have only one help type, like cleaning, babysitting, etc. Each task has an earliest start, a latest finish and duration definition. A Task needs to be performed within its earliest start and latest finish time.

Employee

An employee who works for homecare organization and who is responsible for delivering help to the client. Each employee has specific skills and qualifications.

Employee Contract

Each employee has a contract with the homecare organization which defines the working times and the salary of the employee. In some cases it also defines specific preferences of the employee such as a preferred carrier path, etc.

Work Pattern

The work pattern defines the start time, break times and end time of each working day within a week. Typically a full time employee starts working at 08:00 and takes lunch break at 12:00. Then she starts working again at 12:30 and ends her work at 16:30.

Team

A group of employees who deliver service to specific geographic area. Each employee is assigned to a team.

Area

A group of employees who have similar set of skills and deliver service to the clients located in same the geographical area. Each employee is assigned to a team.

Geographical area boundaries are determined by the company rules. Each LHO has different rules for determining Areas. Most common examples are division based on demand, division based on the employees, etc.

Area Team

A group of employees who have similar set of skills and deliver service to the clients located in same the geographical area. Each employee is assigned to a team.

Short Term Planner

An administrative employee who is responsible for maintaining the plan of one or more Areas. The short term planner aims to maintain the short term plan with minimum changes possible. The most common tasks of short term planner are registering sickness of an employee, replacing her tasks with other employees, etc.

Long Term Planner

An administrative employee who is responsible for maintaining the plan of one or more Areas.The long term planner is more concerned about the optimality of the schedule. The long term planner aims to plan all of the demand in such a way that it meets organizational objectives like client and employee satisfaction, costs, etc.

4.1.2. The organization

LHO employs in the range of hundreds to ten thousands of employees. The employees are grouped into teams who deliver a specific type of service (personal care, psychological support, etc.) to a client at a specific location.

Figure 5: Simplified organizational structure

A simplified LHO organizational structure can be seen in Figure 5. The organization is decomposed hierarchically based on the geographical measures. In our simplified example, the LHO is first decomposed into large regions (e.g. provinces), then to cities and lastly to areas. In practice, the number of hierarchical levels can differ for each company. However, the lowest hierarchy is mostly an Area. Sometimes it may also be called Team or Area Team.

The planning department is also structured to fit into this organizational structure. There is at least one planner for each *Area*. The most common practice is to assign two planners to every *Area*. One of the planners is responsible for short term planning of the *Area* while the second one is responsible for long term planning of the *Area*. The objectives of the both planners differ from each other. The long term planner is more concerned about the optimality of the schedule. On the other hand, the short term planner maintains the already existing plan with minimum changes possible.

4.2. The software requirement of LHO

LHO usually deploys a scheduling software to solve its complex scheduling problem. The scheduling software intends to support decision makers by providing different sorts of information, generating schedules, calculating scheduling outputs, etc.

In this section, we will transform the scheduling requirements of LHO into software requirements. We will consider operational planning and dispatching (see section *4.3 The scope* for details) and we will concentrate only on low-level scheduling processes. When defining the scheduling requirements, we will exclude data interfacing requirements.

The scheduling requirements were defined as a result of the analysis which we did with two LHO (both employ over ten thousand employees) located in North Europe. We did a series of interviews with various stakeholders including the management, planners, finance managers, etc. The requirements are grouped into three categories: functional, non-functional and architectural.

Functional requirements

Functional requirements describe the functions which define what a system is supposed to accomplish.

In this study we are seeking to design a scheduling system which will support low-level scheduling processes (daily scheduling activities, e.g. plan a task). In this section we will transform the low-level process requirements into functional software requirements. We describe the scheduling functionalities in the form of use cases. The list of the use cases that we see relevant for this case can be seen in Appendix 1: Use cases. We summarized the five use cases as four major functional requirements:

Functional Requirement 1 (FR1): The first requirement of the LHO scheduling software is to support low-level scheduling processes, e.g. plan a task, plan holiday, etc. Each use case in "Appendix 1: Use cases" represents a low level scheduling process.

Functional Requirement 2 (FR2): The scheduling system must support the LHO rules, constraints and objectives. Employees, clients and the organization have various objectives and constraints (See *Difficulties in scheduling for LHO* section for details). These objectives and constraints are usually transformed to *scheduling rules*. These rules can have different level of aggregation. The scheduling rules are also incorporated to the use cases defined in Appendix 1: Use cases.

Functional Requirement 3 (FR3): It must support different aggregation levels. When making a decision, the planner must check policies at different levels. For instance, when planning a holiday for one of the employees, the planner must check if the employee is authorized to have a holiday. This check can be considered as *local check* because it only concerns a specific employee. At the same time, the planner must check the capacity of the region or even the city and compare it with the available workforce before approving a holiday. If the available workforce cannot fulfil the demand, then the planner may choose not to approve the holiday.

Functional Requirement 4 (FR4): The software must support resource sharing between different *Areas*. The resource sharing requirement is defined by Use Case 5 in Appendix 1: Use cases.

Non-functional requirements

Non-functional requirements specify **how** a system is supposed to function. The non-functional requirements of LHO scheduling system are defined below:

Non-functional Requirement 1 (NR1): It is expected to support distributed decision making since the planners are located at different locations.

Non-functional Requirement 2 (NR2): The software is expected to support multi-user architecture because there can be multiple planners who share responsibilities. The short term planners, for instance, do not have predictable work environment. The organization doesn't know who is going to call next with what request. Therefore they assign similar responsibilities to every maintenance planner (a planner who is responsible for maintaining the current week). In other words, at a given time T, there might be more than one planner editing/reading the same information.

Non-functional Requirement 3 (NR3): The software must protect the feasibility of the plan at all times. In other words, it should not allow conflicting decisions to be taken simultaneously.

Non-functional Requirement 4 (NR4): The software should display the latest status of the planning changes to all related planners. In other words, if the change made by one planner is affecting another planner, the second planner must be notified for the changes that the first planner makes. For instance, consider the case where employee works for 3 days in one *Area* and 2 days in another *Area*. The planning decision for the first 3 days are made by one planner while planning decisions for the last 2 days of the week are made by a second planner. However, the decisions made by the first planner will affect the decision of the second planner since the employee has only one contract. The system must be able to notify the planners if the employee is over-planned.

Non-functional Requirement 5 (NR5): The software must be responsive in order to support fast decision making. We define responsive scheduling system as a scheduling system which responds to schedule change requests without violating the timing constraints. Each use case of the scheduling system will have a timing constraint which defines the expected runtime of the use case.

Architectural requirements

Architectural requirements describe the system architecture constraints. The architectural requirements of LHO scheduling system are defined below:

Architectural Requirement 1 (AR1): The software must comply with the LHO IT infrastructure. There may be various infrastructures deployed throughout LHO but the most common one is centralized IT. In this infrastructure the computing power is located at one central location and the users (planners) are given monitors (thin clients) which they use to monitor their space (every planner can only monitor the sections which she is responsible for).

4.3.The scope

For this study, a *LHO scheduling software system* is defined as a decision support system that can help the planners to perform daily activities such as plan-in operation, plan-out operation, register unavailability, etc. The system also provides support for different types of rules like governmental or organizational regulations, or employee/client preferences, etc.

In this study we do not seek for a solution that will optimally assign tasks to employees. Instead, we seek for a real time system which will support planners to perform daily scheduling activities mentioned in the paragraph above (daily activities are explained in detail in “Appendix 1: Use cases”).

The daily scheduling activities mentioned in the paragraph above are concerning Operational Planning and Dispatching. The other types of planning are excluded from the scope of this project. The scope of Operational Planning and Dispatching are defined as follows:

1. **Operational Planning:** At this level of planning, each task (demand) is assigned to one or more employees. Additionally, holidays or absences of employees are also planned at this level of planning. Usually it covers the next 2-5 weeks.
    a. The functionalities covered in operational planning are described by use cases. The use cases considered in this study are defined in the Appendix . Use cases from 1 to 5 are related to operational planning.
2. **Dispatching**: The Dispatching department is responsible for the maintenance of the plan which was created at an Operational Planning level. The need to change the plan occurs in the case of an employee falling sick or a client going to hospital, etc. The Dispatching department responds to such unexpected events by rescheduling task(s), assigning the employee to new task(s), etc. The changes in this level of planning are done in real-time.
    a. The functionalities covered in operational planning are described by use cases (See the Appendix for the list of use cases). Use Case 1, Use Case 2, Use Case 4 and Use Case 5 are included in Dispatching.

5. Literature Review

The existing literature was studied in four parts. Firstly, we studied the literature related to distributed scheduling system architecture for homecare. Secondly, we searched the literature of related fields involving human resource scheduling in distributed environments. Thirdly, we searched for scheduling software architectures (ontologies) that may be partially applied to a homecare problem. We also included software architectures for distributed manufacturing scheduling systems as the distributed scheduling field is at quite an advanced stage in the manufacturing area. These four parts will now be elaborately discussed.

(I) There are some studies which explored the homecare scheduling area using distributed and/or multi-agents systems. However they failed to include two or more of the seven requirements mentioned in the beginning of this section. The papers that are closely related to our study in this stream of research are:

- ✓ Itabashi et al. (2005) define a decision support system for Home Care services in Japan. They modelled each employee and client as an agent. Each agent functions autonomously and scheduling is achieved by negotiations between agents. This study however doesn't include support for high level aggregation rules. Additionally, the scheduling is done by employees themselves rather than planners.
- ✓ Corchado et al. (2008) developed a multi-agent system named ALZ-MAS which works in real environment. The application itself schedules different tasks using deliberate agents. But it doesn't support multi-user architecture and global policies. It can be considered as an important example for deliberative agent.
- ✓ Fraile et al. (2009) developed a hybrid multi-agent architecture for homecare environments to monitor patients at their houses.
- ✓ Huang et al. (1995) defined agent based approach to Health Care Management. The decision support was designed for situations where distributed and collaborative decision making is required. However, it doesn't include detailed homecare constraints and lacks support for global policies (global autonomy).
- ✓ Date and Matsuo (2008) proposed a method of discovering the best combination between helpers and elders under much restriction based on multi-agent systems. The agent system was based on negotiation among users. The organizational structure and autonomy defined in this study was different than our problem.

A number of papers in the homecare literature concentrate on optimal scheduling of the homecare organizations. They usually also incorporate soft scheduling constraints so as to handle human scheduling problems. This approach is helpful for our problem since it may reduce the need for changing the plan by improving the initial plan. However, it will never remove the requirement for real time system which handles daily schedule changes that requires distributed, multi-user handling. The models that are most relevant to our study are:

- ✓ Eveborn et al. (2006) focused on staff planning problem. They included some restrictions and soft objectives in the model. The model itself however concentrates on one time solution of the planning puzzle.
- ✓ Borsani et al. (2006) defined an optimization model for home care scheduling. The model was defined to solve the planning puzzle for one week.
- ✓ Bricon-Souf et al. (2005) worked on the definition of a coordination platform for homecare. They mainly analysed inefficiencies due to lack of coordination and proposed a method to avoid these inefficiencies.

- ✓ Hutzschenreuter et al. (2008) proposed an agent-based simulation model for admission scheduling in hospitals.
- ✓ Begur et al. (1997) provided solutions to the assignment problem for homecare business. However, these methodologies are not suitable for real time systems with timing constraints because solution times are longer.

(II) In the literature there are also examples that are related to other fields than homecare but provide a useful approach to multi-user, decentralized systems which support soft timing constraints and global policies. These studies are not adequate to solve our problem but they might provide useful insight for different methodologies:

- ✓ Israel and Heineman (1996) developed a decentralized atomicity model for multi-site, decentralized workflow management system. The study itself is far from homecare environment but it is a good example for a procedure which tries to protect the consistency of data in the decentralized environments.
- ✓ Carrascosa et al. (2008) represented a multi-agent architecture for real-time problems. The study focused on hard timing constraints and it is not in the homecare domain. Nevertheless it is an important example where deliberative and reactive processes are included together.
- ✓ Isern et al. (2010) created a detailed review for the agents which are applied in the health care domain. They concluded that agent technology can offer value to planning and resource allocation in health care domain.
- ✓ Ben-Shaul et al. (1992) proposed an architecture for multi-user software development environments. Although it lacks some parts of the LHO scheduling requirements, it can serve as a good base for data sharing purposes.
- ✓ Huang et al. (1994) defined a corporation method for physically distributed health care settings. They focused on developing communication standards and agent commitment.
- ✓ Koutkias et al.(2005) proposed a multi-agent system for the management of chronic diseases. In their model, they included an approach to minimize the communication between agents by replicating the previous state of agents to the Information Blackboard.
- ✓ Nealon and Moreno (2003) argued that multi-agent systems are beneficial to health care solutions (including home care) because of their autonomous and dynamic behaviour.
- ✓ Pinelle and Gutwin (2001) defined collaboration and communication requirements for Home Care which are interesting for agent based modelling.

(III) Several studies in the literature have proposed ontologies and planning approaches for various industries. Although these studies were defined for industries other than homecare, there are lot of elements that can be reused for homecare environment. The ontologies in the literature are as follows:

- ✓ Rajpathak et al. (2001) presented generic task ontology for scheduling problems. The study covers a lot of aspects as scheduling objects, hard/soft constraints, preferences, etc. However, it lacks LHO specific aspects as organizational structure, distributed decision making, etc.
- ✓ Motta et al. (2002) proposed a general ontology for all types of scheduling problems however it lacks sector specific requirements as company structure, multi-user support, etc.
- ✓ Frankovič et al. (2002) defined a general guideline for creating scheduling and planning ontology.

- ✓ Merdan et al. (2007) proposed an ontology for assembly domains.
- ✓ Wang (2010) proposed an approach for developing a distributed process planning, real-time monitoring and remote machining system.
- ✓ Guo and Zhang (2009) proposed a multi-agent method for dynamic and flexible manufacturing scheduling which is based on cooperation and coordination among the agents.

Literature categorization

In order to systematically compare the relevant materials in the literature, we categorized the studies in the literature. Based on the software requirements defined in section 2.3 (The software requirement of LHO) we derived seven categories. The list of the categories and mapping between categories and software requirements are given below:

- **Decentralized** [D]: decision making is dispersed to smaller regions. (NR1)
- **Multi-User Support** [MUS]: support simultaneous decision making from different actors in cases where decisions interfere with each other. (NR4, NR2)
- **Soft Timing Constraints** [STC]: soft timing constraints are crucial to achieve responsiveness. (NR5)
- **Local Decision Making** [LDM]: most of the times, the planning decisions are local. The planner must be able to work independently. (NR5)
- **Global Policies** [GP]: The system is expected to support common global polices like exchanging employees, trainings schedules, holiday management per location, etc. (FR2, FR3, FR4, NR3)
- **Home/Health Care** [HC]: scheduling of human resources is very different than machine scheduling as people have much more states than functional and non-functional. Furthermore homecare has its special environment where the employees are delivering services at the home of the client. (AR1, FR1)
- **Low-Level Process Management** [LLPM]: due to the environment, there are lots of uncertainties involved in the planning process of the homecare organizations. Therefore the generated schedules must be frequently updated. (FR1)

The current literature does not provide any solution that covers all seven simultaneously. Most of the available solutions cover the items separately or covers only a part of the requirements. The categorization of the available literature can be found in Table 4.

Table 4: Categorization of the available literature [as of 18th June, 2010]

Reference	D	MUS	STC	LDM	GP	HC	LLPM
{A. Moreno, 2004}	☑	☐	☐	☑	☐	☑	☑
{Akjiratikarl, 2007}	☐	☐	☐	☑	☑	☑	☐
{Berrada, 1996}	☐	☐	☐	☑	☑	☑	☐
{Borsani, 2006}	☐	☐	☐	☑	☑	☑	☐
{Carrascosa, 2008}	☐	☐	☑	☑	☐	☐	☑
{Corchado, 2008}	☑	☐	☑	☑	☐	☑	☑
{Ernst, 2004}	☐	☐	☐	☑	☑	☑	☐
{Eveborn, 2006}	☐	☐	☐	☑	☑	☑	☐
{Fraile, 2009}	☑	☑	☑	☑	☐	☐	☑
{Graham, 2001}	☑	☐	☑	☑	☐	☐	☑
{Huang, 1994}	☑	☐	☑	☑	☐	☑	☐
{Huang, 1995}	☑	☐	☑	☑	☐	☑	☐
{Israel, 1992}	☑	☑	☐	☑	☐	☐	☑
{Israel, 1996}	☑	☑	☐	☑	☑	☐	☑
{Itabashi, 2005}	☑	☐	☐	☑	☐	☑	☑
{Koutkias VG, 2005}	☑	☐	☐	☑	☑	☐	☑
{Mackworth, 2003}	☑	☐	☐	☑	☑	☐	☑
{Neelamkavil, 1992}	☐	☑	☐	☑	☐	☐	☑
{S. Begur, 1997}	☐	☐	☐	☑	☑	☑	☐

6. Object Model

In this chapter, we will define an object model that is suitable for the LHO environment. As the scheduling problem of LHO is too large to be managed as a single problem, we first look for a method to simplify the scheduling problem by separating the whole problem into independent sub problems. Then we will then subsequently define an object model for the independent separations.

After defining a method to separate the scheduling system into separate parts, findings from the existing object models in the literature will be presented. Thereafter, we will present the object model through the use of class diagrams. Afterwards we will demonstrate the use of the object model in a distributed, multi-user environment.

6.1. Scheduling system separation

The focus of our problem is on large homecare organizations. As the name suggests, these homecare organizations are large in size, sometimes reaching tens of thousands of employees. In order to manage the schedules of such large organization effectively and increase their responsiveness, we propose to separate the scheduling problem into several independent sub-problems. Each separation will be independent from other separations and each separation will have its own scheduling system.

The most preferred way of separating the scheduling problem will be to decompose the scheduling problem into very small scheduling problems in such a way that every decision maker can work independently. But the LHO scheduling software cannot be decomposed to very small pieces which concern only one decision maker because of two reasons: (i) resource sharing among locations (ii) planners have to base their decisions on aggregate level counters which can be influenced by other planners (FR3 and FR4).

In the literature, we can find several types of separations, mostly defined for multi-agent systems. Most of them divide the problem into autonomous agents. However, these solutions are generally far from being responsive and they lack the support for global policies (global autonomy). For instance, Itabashi et al. (2005) created a decision support system for Home Care services in Japan. They modelled each employee and client as an agent, thus giving autonomy to clients and employees, instead of the planners. However, this kind of separation doesn't include support for high level aggregation rules. Additionally, the scheduling is done by employees themselves rather than planners. Similarly, Huang et al. (1995) defined agent based model for situations where distributed and collaborative decision making is required. However, it doesn't include detailed homecare constraints and it is lacking support for global policies (global autonomy).

During our research, two major factors were identified that have influence on schedule separation: these are *shared resources* and *aggregate counters* on higher organizational levels (e.g. available capacity per Area, per Region, per City, per Province, etc.). Consequently we based the separation of the scheduling method on these factors. The scheduling system can be separated with the following steps:

1. Identify the organizational or practical rules which constrain the resource sharing. The lowest organizational unit that fits to resource scheduling rules can be used as independent separation. For instance, sharing resources is only meaningful when two Areas are close to each other. An Area which looks for additional capacity will hardly request capacity from another province because

travelling time will be very long. In this case, a province will be the smallest organizational unit to be used as a separation unit.

2. Identify the aggregation levels which are influential for decision making for the schedulers. Take the highest level as the smallest separation unit in order to handle different aggregation rules. For instance, the available capacity rules are commonly checked for the City or the Area but not for the Province.
3. Take the larger separation unit of the first two steps and use it to separate on the scheduling problem into smaller pieces.

Every separation will act as a separate scheduling system. Hence there will be no physical dependency between planners who are not influencing each other. For instance, the installations can be done on different computers to remove CPU dependencies, memory dependencies, etc. Removing dependencies will increase the responsiveness (NR5) of the scheduling software as there will be less interrelated users.

6.2.Relevant Ontologies in the literature

The existing ontologies in the literature do not cover the homecare process. The existing approaches focus mostly on the manufacturing processes. Motta et al. (2002) proposed a general ontology for all types of scheduling problems. However, it lacks sector specific requirements as company structure and multi-user support. Frankovič et al. (2002) defined a general guideline for creating a scheduling and planning ontology. Merdan et al. (2007) proposed an ontology for assembly domains. In the Class Diagram section we evaluate further the ontologies in the literature and describe the overlaps of these ontologies with our research.

6.3.Class Diagram

This section will describe an object model for LHO scheduling software system. The model will be presented using an UML Class Diagram.

This section is organized as follows. We start with the definition of classes that are used to model the LHO organization like employee, clients, contracts, organizational structure, etc. Next, we describe the classes that are used to meet scheduling requirements. After this, we describe classes (and processes) that are used to support business rules and resource sharing. The last part of this section will describe the classes that are used to a support distributed, multi-user architecture.

6.3.1. Organizational Classes

This part will describe the classes and their relations that are used to model the LHO organization. We divided these classes into three parts: Employee, Clients and Organizational Structure classes respectively.

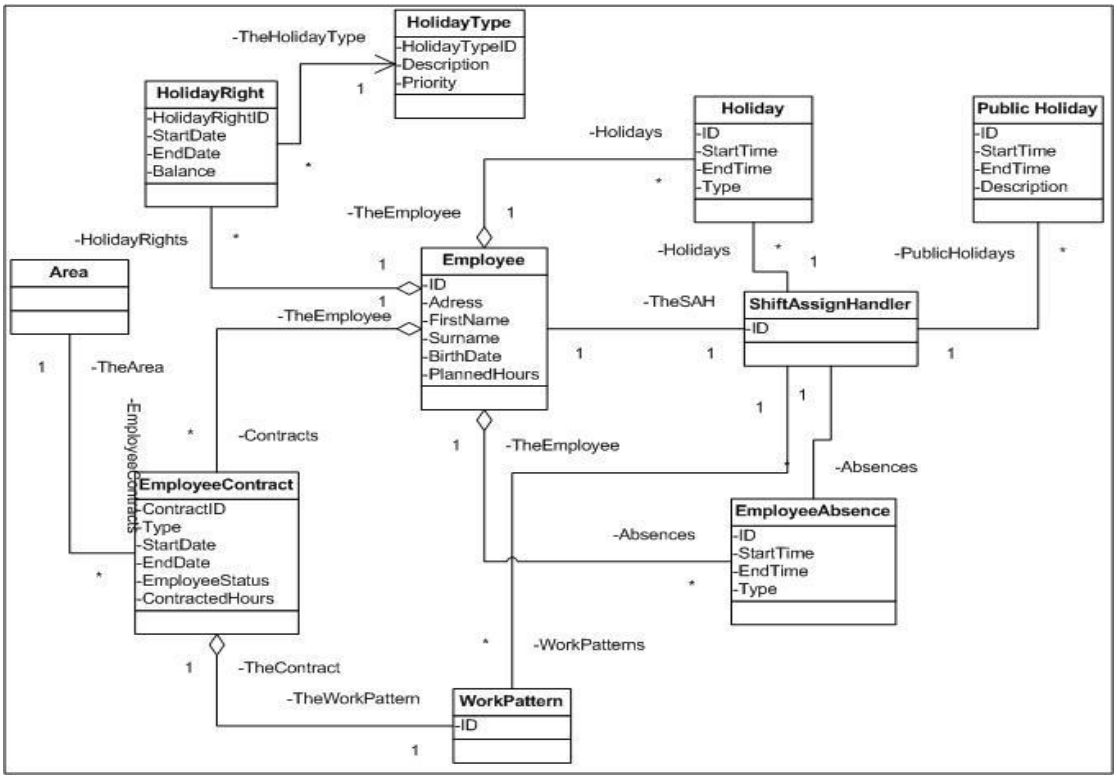
Employee Classes

Figure 6 shows the class diagram which is used to model employee related attributes. Rajpathak et. al. (2001) generalizes the employees as resources. In our architecture, the employees of the organization are modelled by an *Employee* class. In our architecture, we consider only Employees as scheduling resources. Every employee has one or more contracts with the organization. Contracts are represented by a *EmployeeContract* class. Every contract also defines the *WorkPattern* of the employee.

The *ShiftAssignHandler* defines the time intervals when an employee is working. The *ShiftAssignHandler* calculates the working times by subtracting the *Holiday*s, *Absence*s and *PublicHoliday*s from the *WorkPattern* of the employee.

Every employee has a right to have a holiday every year. Her holiday allowance (or initial balance) depends on her contract, age and her history with the company. The holiday right is modelled by a *HolidayRight* class. Every holiday right has a type, modelled as *HolidayType*.

Figure 6: Employee related classes



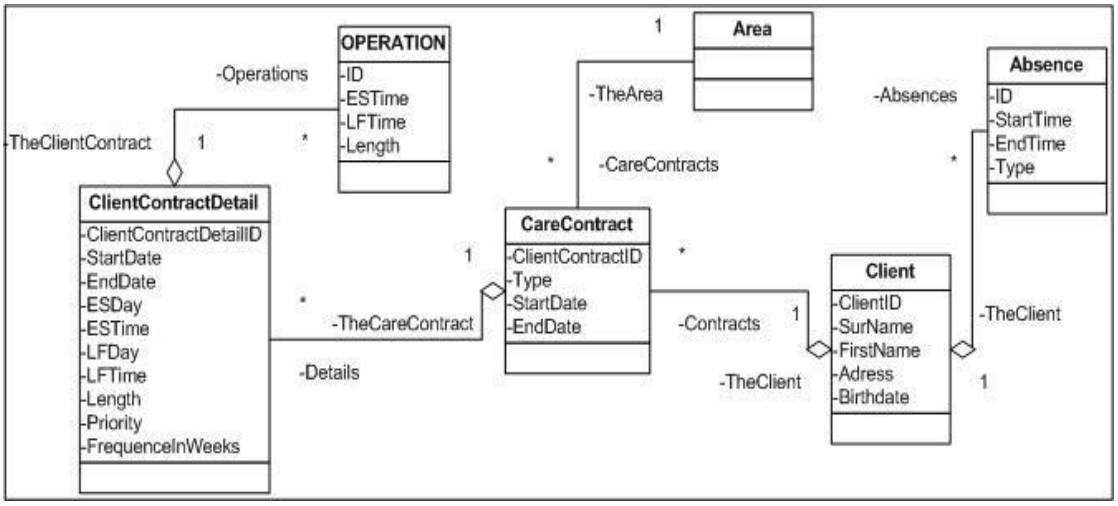
Client Classes

Figure 7 shows the client related class diagrams. The client class also defines the service requirements. Rajpathak et al. (2001) models clients as an attribute of job. Frankovič et al. , (2002) models clients as a separate class named “customer”. In our model, we consider a client as separate object.

Each *Client* has one or more contracts with the organization. A clients’ contract(s) with the organization is modelled as *CareContract*. Each care contract contains details about the work that needs to be performed and the time frame it needs to be performed in. These details are modelled through a *ClientContractDetail* class. The tasks are generated based on the *ClientContractDetail.* The tasks are represented with a *Operation* class.

For LHO case, client needs to be available during service delivery. Therefore we model the client absences as *Absence*.

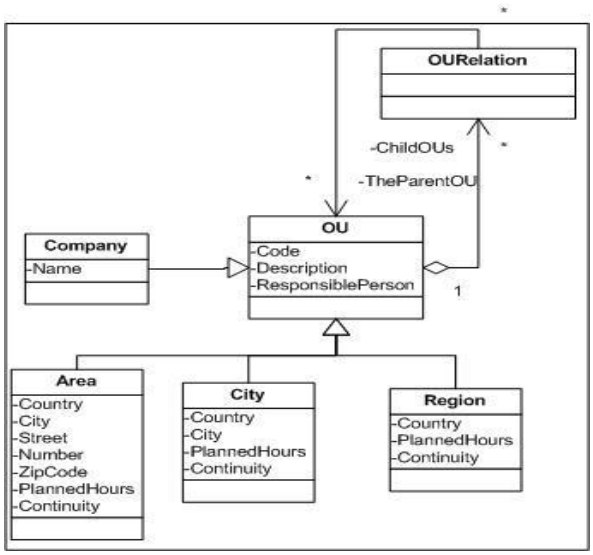
Figure 7: Client related classes



Organizational Structure Classes

The organizational structure varies from organization to organization. Factors like size, geographical location are influential on the structure of the company. Figure 8 represents the organizational structure model which we will use in this study. In our research we assume that the organization deploys a team based structure. Therefore the basis for the organizational structure is *Area*. The *Area* class in this model represents a team of employees who are working in a specific geographical region. Each employee is assigned to at least one *Area*.

Figure 8: Organizational structure classes



The organizational levels of the company are generalized as an *OU* (organizational unit) class. An OU represents the hierarchies in the organization. Every organizational level may have more than one child. The organizational hierarchy is modelled by *OURelation* class. This way every *OU* knows her parent *OU* and vice versa.

In this study, we will use *Area*, *City*, *Region*, and *Company* as organizational units. The *Company* is decomposed into several *Regions.* Every *Region* contains one or more *Cities*. Each *City* contains several *Areas*.

The three levels (*Area*, *City*, *Region*) were chosen as representative set of hierarchy. The planning decisions are made on an Area level as the planner responsibilities are defined per Area. Therefore we choose higher hierarchical levels in the organization. We consider these three levels as sufficient in order to demonstrate aggregated counters.

6.3.2. Scheduling Classes

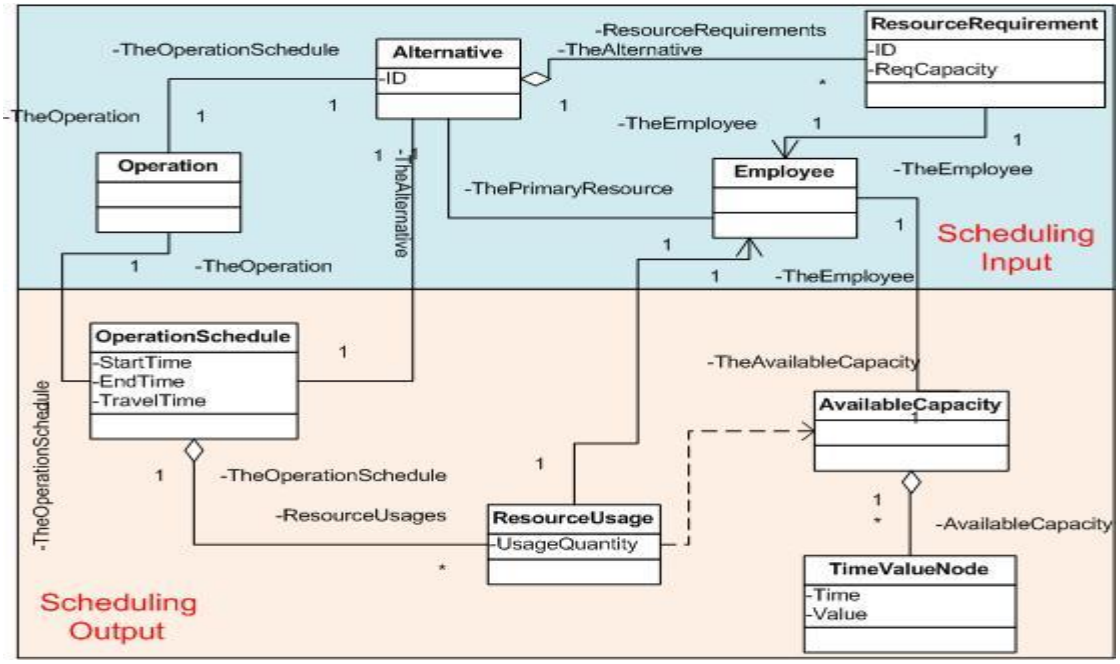
The scheduling classes can be categorized into two parts: scheduling input classes and scheduling output classes. The scheduling input classes represent the calculated information that is extracted from different sources. The scheduling output classes represent the classes which are used to model the outputs of the scheduling process, e.g. the task assignments, capacity usages, etc. In Figure 9, the scheduling input classes are shown in the blue rectangle while the scheduling output classes are shown in the pink rectangle.

The scheduling input classes are usually derived from information which is extracted from different sources. *Alternative* class represents the alternative employees who can perform the *Operation*. An *Employee* can be an *Alternative* for an *Operation* if:

- The *Employee* who has the necessary skills to perform the task; and
- The *Employee* lives in the same city as the *Client; and*
- The *Employee* who fits with the other business rules.

A task may require one or more *Employees*. The requirement is represented by *ResourceRequirement* class.

Figure 9: Scheduling classes



Scheduling is done by assigning operations to employees. The scheduling information is represented by a class named *OperationSchedule*. The *OperationSchedule* object keeps information about the used resources. The usage of resources is represented by *ResourceUsage* class. Resource usages are used by *AvailableCapacity* to calculate the net available capacity at each moment. *AvailableCapacity* has a list of *TimeValueNode*'s which shows the increment/decrement times and the value (e.g. capacity) at that time.

6.3.3. Rule Support Classes

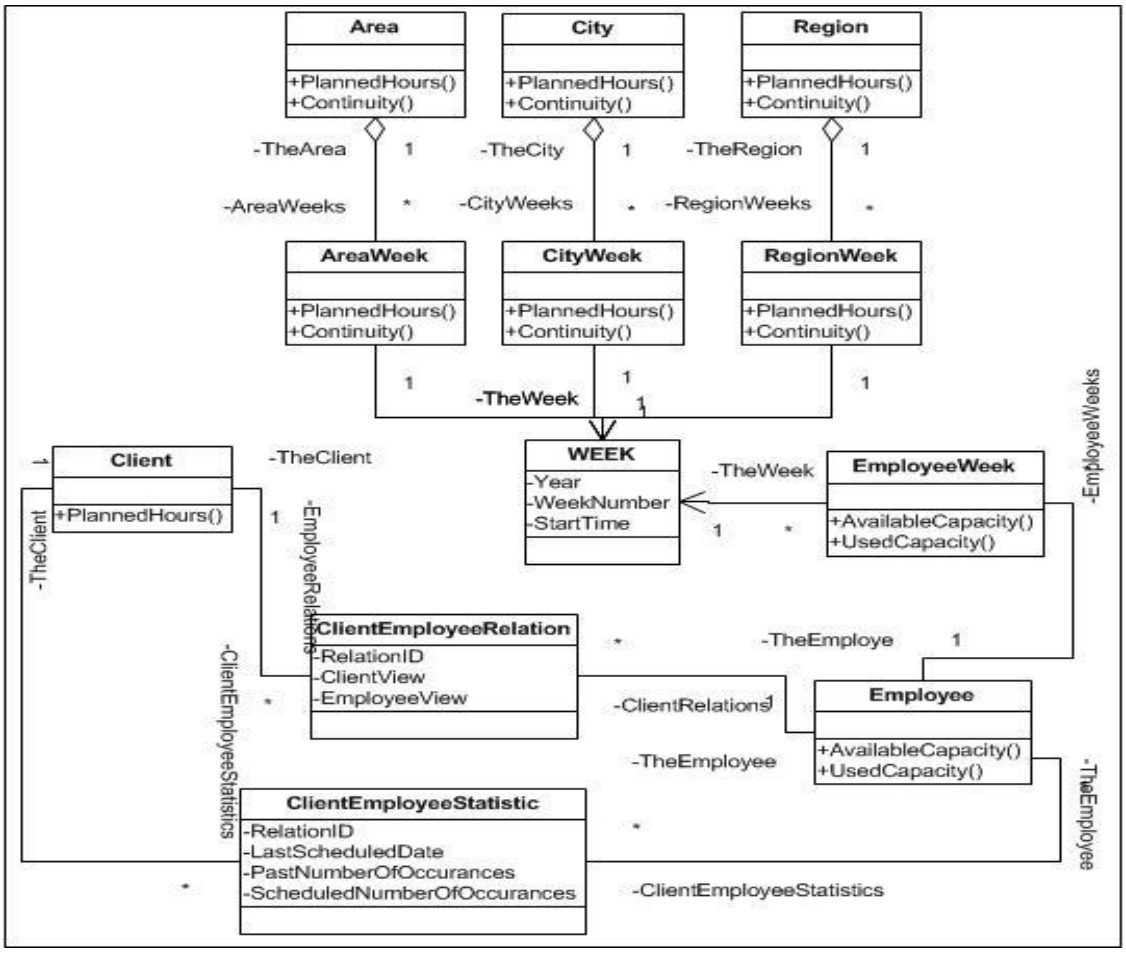
Homecare organizations have different indicators at different levels of aggregation. We refer to them as counters. The counters are used to monitor organizational policies. For instance, LHO want to keep track of the "total planned hours" for every *Employee*, *Area*, *City*, *Region, etc*. In this way the organization can control the planned capacity and compare it with the demand. The "planned hours" indicator needs to be calculated per week so that it can be compared to contracted hours (contracted hours of employee are defined on a weekly level).

In addition to the aggregated values, there are also some variables on the employee or client level. For instance, it is important to know the capacity utilization on the employee level, the fulfilled hours for each client as well as the number of visits the employee makes to each client.

Figure 10 shows the classes which are used to model rule checks.

The Rule Support classes that we consider in this section are more representative that complete the set of counters. With this set of counters, we aim to demonstrate the different level of aggregated counters. Most of the scheduling decisions are made on counters on Area level or employee/client level. However, some decisions are made on higher level counters. In order to include such decisions in the architecture, we also include City or Region level counters. Most frequently, counters are measured for a specified time frame. We choose WEEK as a representative time period which we will use in our architecture.

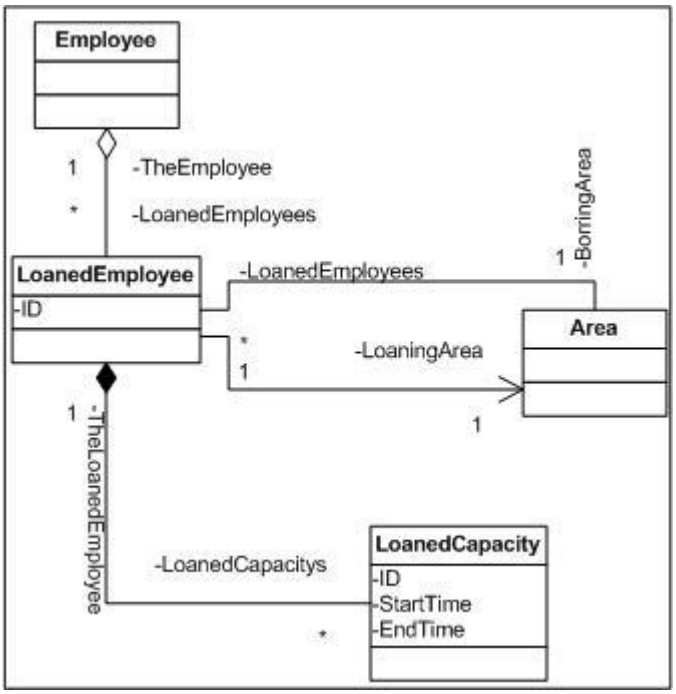
Figure 10: Rule support classes



6.3.4. Shared resource classes

Due to the uncertain nature of the homecare environment, it is common to share resources between *Areas*. This process is modelled as loaning an *Employee*. Figure 11 shows the classes which are used to model resource sharing between *Areas*. A *LoanedEmployee* class is used as an intermediate between two *Areas*. The *LoanedEmployee* class represents an *Employee* (TheEmployee) who is loaned from one *Area* (LoaningArea) to another *Area* (BorrowingArea). An Employee can be loaned multiple times from one *Area* to another at different times. *LoanedCapacities* show the different time intervals during which an employee is loaned between *Areas*.

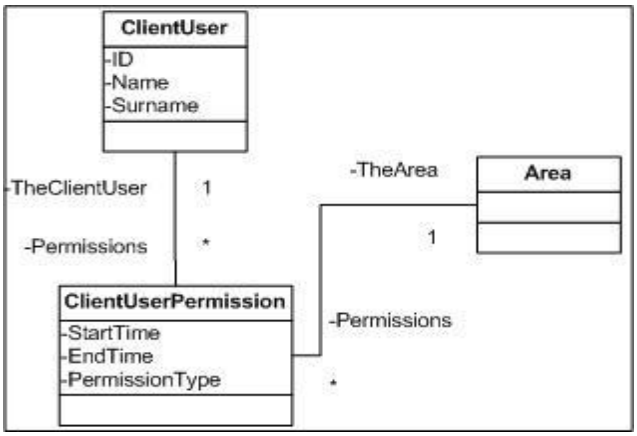
Figure 11: Classes related to shared resources



6.3.5. Distributed Decision Making Support Classes

The LHO scheduling system will be used by multiple users that are working at different geographical locations. Every planner will have different responsibilities with respect to different *Areas*. Some planners will be responsible for short term planning of the *Area*; others will be responsible for long term planning. In addition, there might be planers who want only to see the planning (read only). The scheduling system needs to support an authorization mechanism which will set the boundaries of each planner.

Figure 12: Distributed Decision Making Support Classes



The authorization mechanism is based on *Area*. Every planner can be authorized to change or read the parts of the schedule which are related to a specific *Area*. Figure 12 represents the classes and their relations that are used to model the authorization mechanism. Each planner is modelled as a *ClientUser*. Every *ClientUser* has one or more permissions (*ClientUserPermission*) to one or more *Areas*. The permissions are defined for a specific time window. Outside that window the permission is invalid.

The next section will explain the authorization process in detail.

6.4.The distributed decision making and authorization process

The authorization mechanism is required for a LHO scheduling system because the users are distributed over wide geographical regions and cannot communicate with each other directly. As a result, their will to change the planning must be regulated in such a way that their decisions are not conflicting with each other.

In order to avoid conflicts in the schedule, we introduced the authorization mechanism. An authorization mechanism restricts a planner in terms of geographical region and time. In other words, a planner can edit the schedule of an Area only if she has the right to change it.

Moreover, we also restricted the number of users who can edit the planning of an Area at the same time as it is not very meaningful to allow different planners to change the planning of the same employee for the same day. We allow only one Long Term Planner (LTP) and one Short Term Planner (STP) to edit the planning of an Area at any time T. Before starting to plan, the planner sets the Area she wants to work on as her current Area and sets her role (LTP or STP). If there is already a user who is working on the same area with the same role, then a second user is not allowed to edit the schedule of the Area.

The authorization time window of a user doesn't only depend on the rights of the user but also on the planning status. The LTP is responsible to create the schedule for the coming weeks. Once she is finished with scheduling, she releases a plan to STP. Then STP gets the authorization for the released period. If the LTP needs to correct the released plan, then STP gives her authorization back to LTP. In such a case, LTP has to release the plan again.

To sum up, we introduce an authorization mechanism which will help to keep the schedule consistent. This way we can allow multiple (LTP and STP) and geographically dispersed planners to work simultaneously on the same scheduling system.

However, the authorization mechanism is not adequate to keep the schedule consistent at all times. There are calculated values on different aggregation levels which can be influenced by multiple decision makers. In such a case, these values need to be protected from concurrent access as concurrent access may cause conflicts. The next chapter will focus on handling issues of objects/ fields/ functions in LHO scheduling system and describe a method to solve it.

7. Schedule consistency and process management

Our goal is to find a suitable software architecture for large and dynamic homecare organizations to achieve a responsive and consistent scheduling process in a distributed, multi-user environment. In the previous section, we defined the first part of the architecture, i.e. the object model. The next step will be to define a systematic method to manage low-level processes in a distributed, multi-user environment. The challenge is to develop a systematic method which will keep the schedule consistent and still be responsive.

In this chapter we will begin with the definition of the consistency requirement and its importance. Then, we will evaluate alternative methods which we can use for keeping the schedule consistent. Next, we will select an alternative and we will explain physical architecture of the selected alternative. Afterwards, we will explain how we will apply the chosen method to create responsive scheduling system. In the final section, we will have a look at the responsiveness when there are software tasks with long durations.

7.1. Importance of schedule (data) consistency

One of the requirements of LHO scheduling system is to achieve consistent scheduling process (NR3). Inconsistencies may occur when two decisions interfere with each other. Most frequently conflicts occur when multiple planners execute decisions simultaneously. For example, conflict occurs when one planner is registering a holiday and at the same time another planner is assigning a task to the same employee for an overlapping time period. If we translate this situation to scheduling software language, it means that two threads are updating the same fields, objects, etc. as both threads are updating data which is related to the same employee object.

Simultaneous access to the same object is not desirable because allows creation of infeasible schedules. In our example, the second planner probably wouldn't have assigned a task to the employee if she would have known that the employee is on holiday for that time period. Such access may lead to a number of problems, e.g. miscalculation or system crash if one of the threads deletes an object when another thread tries to access it. Consequently, we have to protect the data or synchronize access to data (Ben-Shaul et al., 1992) if we want to keep it consistent.

Next section will define the alternative methods to synchronize simultaneous access to data in distributed, multi-user homecare environment.

7.2. Alternative data protection methods

According to (Ben-Shaul et al., 1992), there are two alternative methods to synchronize simultaneous access to scheduling data. First method is to replicate the data in such a way that every user has their own workspace. The second method is to define access rules for the data which is located on central location where every user is connected to.

Data replication

The first method implies that first the data is replicated to each user's space. Afterwards, every change the user makes is only done in her own copy. The last step of the process is integration of the separate workspaces.

Replication of data is good option to reach responsiveness because it allows users to work on their own workspace without interfering with any other decision maker (or being interfered by other decision makers). However, the integration of the separate workspaces is not very easy when there are conflicting decisions. For instance, imagine a case where one

planner is deleting one of the previously created absences and the second planner prolongs the same absence. The scheduling system must be capable of handling such a situation. In practice, it is very hard to define a rule which states priority of one scheduling decision over another, especially if those decisions are made by different planners.

Centralized approach

The second method requires a central architecture. The software (the scheduling system) is placed on a central location (server) and the planners are connected to it via client software. In this architecture, the server acts as a service provider. In our case, the service is scheduling process. The software client is an interface between the planner and the server. It is responsible for displaying scheduling charts, reports, etc. to the planner and initiating scheduling requests. Whenever planner wants to make a change in the schedule, she will trigger the *Use Case* from the client software GUI. The client software will trigger the scheduling activity on the server. The server will process the request and then send the results to the client software. Client software will display the results to the planner.

The centralized approach requires implementation of data locks in order to protect data (or synchronization). The locking mechanism prevents concurrent access to the same object. Instead, simultaneous requests are performed sequentially. This way, the schedule is always consistent.

The biggest challenge in the data locking is to find the right level of locks. If the data lock is defined on very high level, i.e. data subset is very large, then the responsiveness becomes an issue in multi-user applications. On the other hand, lower level locks increases the possibility of deadlock and reduce the maintainability of the application.

During our research, we discovered that the data protection can also be delegated to the database. If the scheduling system was built as database application which displays information from database instead of running on memory, then the data protection could have been delegated to the database. The databases have well-structured synchronization mechanisms for multi-user environments. However, in our case, we are not only displaying information but also processing information. Sometimes the scheduling algorithms require implementation of complex optimization methods or scheduling heuristics. Implementing these procedures on a transactional system will reduce the responsiveness dramatically. Therefore we decided to eliminate this alternative.

Existing implementations

The implementations existing in the literature mostly focus on multi-agent systems. They try to secure data consistency by separating autonomy over objects. They allow only one agent to change the state of an object which prevents inconsistencies. However, such an approach creates problem in response times. Isern et al. (2010) created a detailed review for the agents which are applied in the health care domain. Huang et al. (1995) defined agent based model for situations where distributed and collaborative decision making is required. Huang et al. (1994) defined a corporation method for physically distributed health care settings. They focused on developing communication standards and agent commitment.

There are also good examples for centralized systems from other research areas. Israel and Heineman (1996) developed a decentralized atomicity model for multi-site, decentralized workflow management system. The study itself is far from homecare environment but it is a good example for a procedure which tries to protect the consistency of data in decentralized environments. Ben-Shaul et al. (1992) proposed an architecture for multi-user software

development environments. Although it lacks some parts of the LHO scheduling requirements, it can serve as a good base for data sharing purposes as it provides a good approach to handle multi user software tasks.

There are also detailed studies about Real-Time Database Systems (RTDBS). Likewise LHO scheduling system of our study, RTDBS has to process transactions triggered by different sources and guarantee that database consistency is not violated (Abbott & Garcia-Molina, 1992). Abbott & Garcia-Molina (1992) proposes four component scheduling algorithm for database transactions: (i) a policy to manage overloads, (ii) a policy for assigning priorities to tasks, (iii) a concurrency control mechanism, (iv) a policy for scheduling I/O requests. Abbott & Garcia-Molina (1992) test different scheduling policies for different scenarios like high-load times, increasing conflicts, etc. Even though their study provides very useful insights in transaction scheduling, the transactions they are considering are simpler than the transactions in homecare scheduling. LHO scheduling system requires more comprehensive concurrency control mechanism as the subset to be locked is depending on the content of the use case.

Weikum (1991) studied RTDBS system with layered architecture. He focused on concurrency control strategies. Further he investigated protocols for aborting transactions and restarting the system after crash.

Baccouche (2005) defined a dynamic admission control and parametrable priority based scheduling algorithm called H/M/L.

Haritsa et al. (1991) developed priority assignment algorithm called Adaptive Earliest Deadline which detects overload conditions and modifies transaction assignments accordingly.

Weikum and Hasse (1993) developed algorithms for multi-level transaction management in a database kernel system.

Weikum and Schek (1992) studied concepts and applications of multi-level transactions and open nested transactions.

Ramamritham (1993) and Ulusoy and Belford (1993) studied concurrency control protocols for RTDBS.

The next session will elaborate the alternatives and choose the one that suits best to our problem.

7.3.Why centralized model for LHO?

There are two different approaches that we can use for synchronization of data: by centralizing the system and controlling the simultaneous access or by decentralizing the system and later integrating the separate pieces into one schedule Ben-Shaul et al. (1992).

Both methods can contribute to our problem. The decentralized approach can be selected because it is more responsive. But it is hard to maintain the consistency of the schedule. The centralized approach will keep the schedule consistent but achieving responsiveness in multi-user environments is challenging.

In our model, we choose the centralized approach because responsiveness and consistency are not the only requirements we are considering. The scheduling system

requires support for different level of aggregation and rules (FR2 and FR3). It is hard to maintain the high level aggregation in a distributed system.

Additionally, the scheduling system must allow resource sharing (FR4). In a decentralized system, implementing the resource sharing use cases will be very difficult due to extensive communication requirements.

Lastly, the LHO usually have centrally focused IT infrastructure. The LHO is spread over large geographical areas. It is not economically feasible for each area to keep their own IT infrastructure. Therefore LHO deploys centralized architectures where computational power is located at central location.

The next section will describe the application of centralized software system to LHO.

7.4.System Configuration

The configuration that we choose for LHO is called Client-Server configuration because LHO IT systems are located on a central location while the planners are located at various locations. Figure 13 shows graphically the physical architecture of the system and the communication between its independent parts. The scheduling system is placed on a central server and the planners are connected to it via client software. In this architecture, the server acts as a scheduling service provider. The software client is an interface between the planner and the server. It is responsible for displaying scheduling charts, reports, etc. to the planner and initiating scheduling requests. Whenever a planner wants to make a change in the schedule, she will trigger the *Use Case* from the client software GUI. The client software triggers the scheduling activity on the server. Then the server processes the request and sends the results to the client software. Client software will display the results to the planner.

There are two software components of this architecture. One is the scheduling engine which is located on the server application. The second one is the GUI component which is located in the client application. LHO scheduling software might contain many other components (e.g. communication manager) but only the Schedule Engine and GUI are relevant for our case.

Figure 13: Client-Server configuration*

*Source: Downloaded fromhttp://code.google.com/edu/parallel/dsd-tutorial.html on 1st of August, 2010

7.5.Implementation of data locking to LHO

This section will propose a data locking method for LHO. First it will describe the basics of data locking. Second, it will define an atomic unit for data locking. Third, it will explain how the data locking is working at LHO environment. In the last part, it will describe a solution for a special case, i.e. concurrent access to the same object.

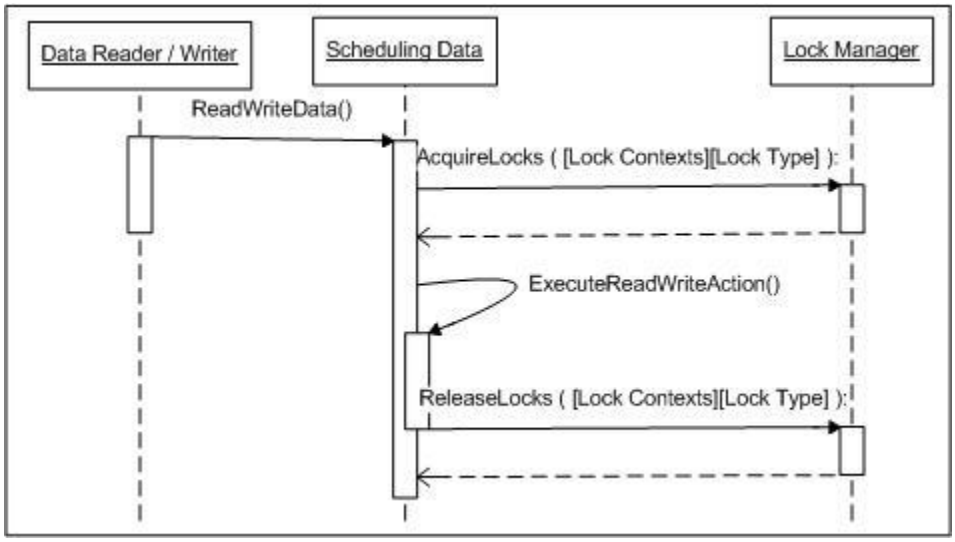
7.5.1. Data locking mechanism

Our goal is to protect the scheduling data. There are two types of access to the scheduling data: **read** and **write**. Software components like GUI are only displaying the scheduling data, i.e. they only need **read** access to data objects. On the other hand, the Scheduling Engine component not only reads data but it also changes, deletes or creates data. Therefore the scheduling engine activities generally require **write** access.

As a result we identified two types of data locks for the scheduling system: Reader Lock (**R**) and Writer Lock (**W**). The both locks control the access to memory which is shared by multiple threads (or users). **R** lock allows concurrent access to multiple threads to the same memory (data). The W lock however is an exclusive lock, i.e. no other thread can read or write to the same memory area once a write lock is acquired.

In order to control concurrent access to the scheduling data, we added Lock Manager component to the LHO scheduling system. The Lock Manager is responsible for granting or denying authorization to threads which are trying to access the scheduling data. When a thread attempts to access the scheduling data, it must first acquire a lock from the lock manager. Figure 14 shows the lock acquisition procedure. Data Reader/ Writer thread requests access to data. Depending on the required access type, the thread requests **W** or **R** lock(s). The thread can start to read or write action after the requested lock(s) has been granted by the Lock Manager. After the thread is finished with reading or writing the data, it releases the acquired locks.

Figure 14: Lock acquisition procedure

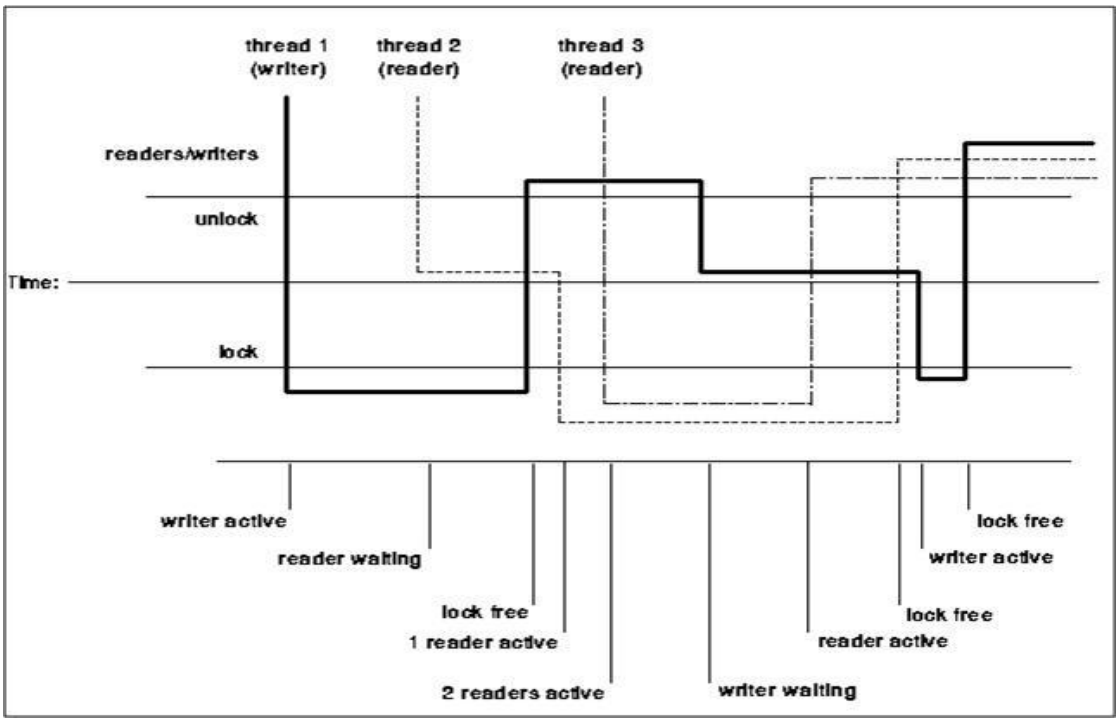


A lock is acquired for specific context. A lock context defines the subset of the scheduling data to be locked. For instance, if we need to lock data related to one employee, then we can define the lock context as the unique ID of the employee.

When there are multiple threads trying to acquire the same lock context, then the lock manager regulates the acquisition process based on prioritization rules. The lock manager will allow multiple threads to have **R** of the same context but it will allow only one thread to be running when there is a thread which has **W** lock. Figure 15 illustrates the behaviour of lock manager with an example of three threads trying to acquire the same lock context.

Additional to **R** and **W** priority rules, we also defined priority rules for the queue. The regular queue is working on FIFO basis. But there is an exception for **W** locks. The **W** locks are more prior to **R** locks and they always go to the front of the queue. The **W** lock has to be given priority because the system allows multiple **R** locks. This means when R lock is taken, the new coming **R** locks will be acquired but **W** lock will keep waiting until all **R** locks are released. Consequently, the priority between **W** and **R** locks need to be implemented to avoid situations where **W** lock waits for very long time.

Figure 15: Lock prioritization



*Source: Downloaded from http://h30097.www3.hp.com/docs/base_doc/DOCUMENTATION/V51B_HTML/ARH9RCTE/DOCU0009.HTM on 1st of August, 2010

The next section will define the lock context which will be used in LHO scheduling software.

7.5.2. The Data Locking unit (atomic unit)

In the previous section, we explained the locking procedure and locking parameters. In this section, we will make the choices regarding the locking parameters, lock context and lock type. In order to describe our choices, we will use thread state diagrams. Therefore we will start this section with definition of states. Then we will continue with our choices. The definition of states is given in Table 5.

Table 5: Thread states

DEFINITIONS
ThreadInitialized Thread is triggered, algorithm is ready for run.

WaitingForLocks

The thread requested locks from the Lock manager and is waiting for reply.

AcquiredLocks

Thread has acquired the requested locks**.**

UserInputsAcquired

Use cases usually requires additional input from a planner. State reaches user inputs acquired state after the system validates the inputs entered by the planner.

RulesValidated

There are various rules/constrants in LHO. These rules are validated before scheduling task starts. A thread is at RulesVallidated state if all rules are obeyed or when a planner accepts the rule violation.

Rules are validated after locks are acquired. Therefore it is not possible for other users to change the state of the objects which are used in rule checks.

SchedulingActionExceuted

A thread reaches this state after scheduling action is executed. For instance after a task is assigned to the employee or task is removed from employee.

Waiting [Master][W] Lock

A thread requested [MASTER][W] lock from the Lock manager and is waiting for it.

Acquired [Master][W] Lock

The thread acquired [MASTER][W] lock from the Lock manager.

AggregateValuesRecalculated

Aggregate values are recalculated after the scheduling activity is executed. Aggregate values are recalculated after [Master][W] lock is acquired in order to avoid concurrent access.

LocksReleased

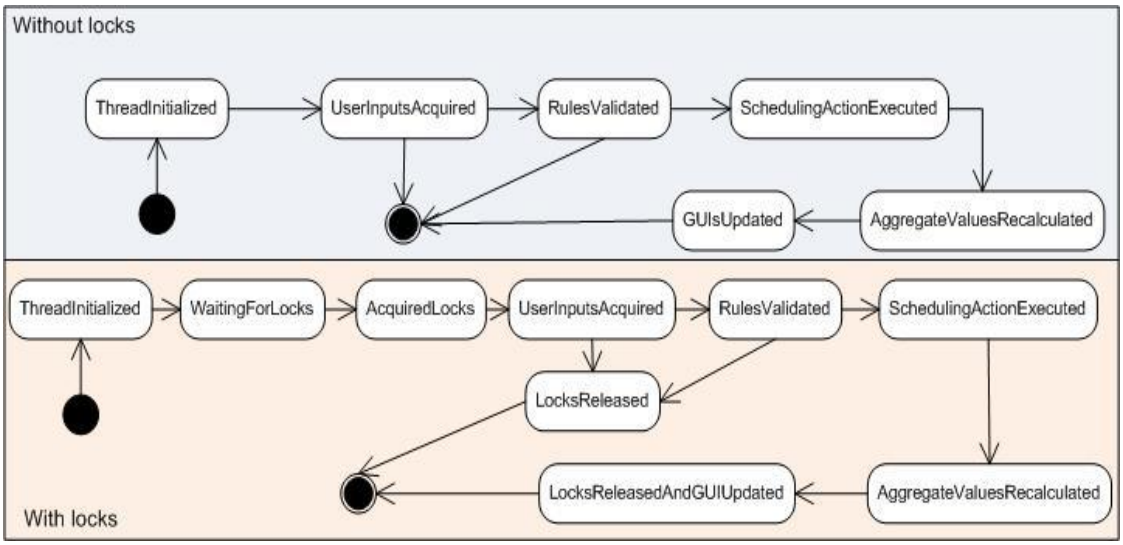
All locks are released.

GUIUpdated

The GUI of the planner and related planners are updated. Related planners definition can vary per organization and per GUI items. Most frequently the GUI update rule is to update the planning GANTT chart of every planner which is working on the Area where the schedule changed.

LHO requires both lock types, i.e. **R** and **W**. The Scheduling Engine component will require **W** locks when executing the use cases. The GUI component will use the **R** lock when retrieving data from the server. Figure 16 demonstrates the states of a thread when executing typical use case of LHO. When executing the use cases of the LHO, it needs to acquire **W** lock because it updates shared information regarding the schedule. On the other hand, the GUI component only needs R lock because it only retrieves the information and displays it.

Figure 16: Thread states (with and without locks)



The second decision regarding the implementation is the lock context. The lock context is a very important concept because it determines the subset of data to be locked. The subset of data to be locked has large impact on responsiveness. The larger the subset, the higher the response time. For instance, if we choose not to split the data and have one subset, then all the software tasks will be running sequentially. Responsiveness will be very poor in such environments. On the other hand, defining a very small data subset as atomic unit brings the complexity of lock management. When locking level is low (e.g. Employee level), then threads require to acquire multiple locks which will increase the chances of running into deadlock situation.

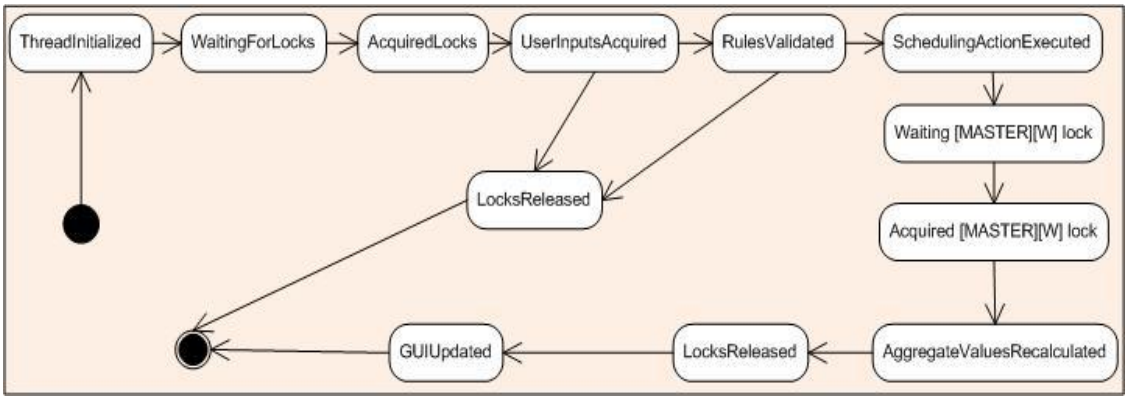
LHO requires a suitable subset which will be easy to maintain and still responsive. We propose that Area (or Team) is a suitable data subset because it is not a very low level subset and it is the only subset that exists in all organizational structures. Additionally, most of the dynamic data of LHO scheduling system is on Area level or lower level. For the software, it means that dynamic object creation and deletion happens on lower levels than Area. Moreover, the management of the locks is easier because every data in the system is connected to Area. Employees belong to an Area, Clients belong to an Area, the planner responsibilities are defined per Area, etc.

The second alternative for data subset could have been employee and client level. However, then the management of the locks will be very difficult because use case itself doesn't define the locks to be required. However, if we define lock contexts on area level, then we can calculate the list of locks to be acquired based on the authorization definition of the user no matter what use case is executed.

Consequently, we can conclude that Area level lock is a suitable for LHO scheduling system. But it is not sufficient subset to protect higher level aggregate values, like Available Capacity per city. Therefore we need to add more locks to protect aggregate values. We identified two alternative locking solutions for higher level aggregate values: (i) to lock all the areas which are below the level (ii) to lock the entire data. As locking of the entire data much easier to manage by a global lock, we preferred to introduce such a lock for entire data. We called this lock MASTER.

Every thread will acquire the MASTER lock as **R** lock when it starts execution. When the execution comes to calculation of higher level values, then the thread will upgrade [**MASTER**][**R**] lock to [**MASTER**][**W**] (the first bracket represents the lock context, the second bracket represents the lock type). After the aggregate values are recalculated, the thread downgrades the global lock to [**MASTER**][**R**]. Since all the threads will have to acquire [**MASTER**][**R**] all the time, they will have to wait if someone acquired [**MASTER**][**W**]. At first glance, it may seem that it will have dramatic influence on responsiveness of the system, but it actually doesn't. The calculation of the aggregate values is done in very short time, in a few milliseconds because all we need to do is to add or subtract a number to/from the existing value. Figure 17 demonstrates the state diagram for a thread with the global lock.

Figure 17: Thread states with global lock



7.5.3. Implementation of data locks

In the previous section we have defined the two type of locks we are using: the area lock and the global lock. The global lock will be used when the thread is calculating aggregate values. The area lock will be used in the remaining part of the thread.

In this section, we will define the method we use to find which area locks to acquire at each thread. Because every thread will be doing a different job, the data subset they will influence will be also different. For instance, if we are assigning a task to of client C to employee E, then we need to locks the Area to which C and E belong. The other Areas doesn't need to be locked since the changes the thread makes will not influence them.

The management of such dynamic lock mechanism, however, requires additional procedure to calculate what locks to acquire at what thread. This procedure can be very complicated because the dataset that needs to be locked depends on the decisions which are done after the execution start of the thread. In order to simplify the management process, we propose to use more pessimistic approach where a thread acquires all the Area locks which user may be influencing.

From section 6.4 we know that a user is authorized to make changes in only one Area at a time. And we are also able to calculate the Areas that may be influenced by a change in one Area. Therefore, we can calculate the list of Areas which the thread needs to acquire based on the user who triggers the thread.

The calculation of the locks to be acquired in each thread will be calculated as follows:

- Every planner is working on one Area at a time. And every Area has a list of Areas which may be influenced by a change in the schedule of the current area. This list is called RelatedAreas. The thread will get the list of the RelatedAreas of the CurrentArea (see section 6.3.5 for class diagram) of the user and lock them. The calculation of the RelatedAreas is done as follows:
    1. Employees of the current area of the user may have contracts with other Areas. The thread needs to lock them to prevent conflicts. Therefore we get the list of Areas for which employees of CurrentArea have contract with. We call it **OA**.
    2. Clients of the current area of the user may have contracts with other Areas. The thread needs to lock them to prevent conflicts. Therefore we get the list of Areas for which the clients of CurrentArea have contract with. We call it **OAC**.
    3. The employees may be loaned to other areas. Therefore we need to get the list of areas to which employees of the Area are loaned to. We call it **LA**.
    4. The area might have borrowed employees from other areas. Therefore we need to get the list of Areas from which the employees are borrowed. We call this list **BA**.
    5. {RelatedAreas} = **{OA}** U **{OAC}** U **{LA}** U **{BA}**

Recalculation of data locks (after changed in Resource Sharing, etc.)

The RelatedAreas list which defines the locks that a thread will acquire before execution is not a static list. It depends on the dynamic factors such as loaned employees or borrowed employees. Therefore the locks to be acquired for a user must be recalculated after a change in loaned employees or borrowed employees, or when employee changes the current area. The recalculation of the RelatedAreas list doesn't only concern the user who makes the changes but also the users who are controlling one of the RelatedAreas. Therefore during the recalculation of locks to be acquired list, the scheduling system must not process any other changes which may cause conflicts. We propose to acquire [**MASTER**][**W**] lock before recalculation of RelatedAreas list and release it after recalculation of RelatedAreas list.

7.5.4. Object Locking

The Area locks are sufficient for concurrency control for most of the scenarios. However there are exceptions where Area lock is insufficient. For instance, if two planners delete the same object exactly at the same time, then the system may be halted. Since both of the planners trigger the delete activity at the same time, both threads will be initialized. But only one of them will be executed and the second one will wait for the locks. When first one is finished, the second one will start execution. However, the object which the second thread is trying to delete is already deleted by the first thread. This situation may have deadly results for the application. To prevent such problems, we propose object locking mechanism which allows only one user to be deleting or editing an object. The object lock manager keeps track

of the locked objects. If an object is locked, the second thread is not initialized. Therefore the problem is cleared.

7.6.Software task prioritization

The key design point of the architecture we propose is the execution time of the Use Cases. Most of the use cases of the LHO scheduling system take relatively short execution time. When there are use cases with long execution times, then the responsiveness become a problem in multi-user environment. To improve responsiveness in situations where long software tasks are executed, we propose to split the long software tasks.

The long software tasks can be grouped into two types. First group is the one which we can break into smaller repeating tasks. The second ones are the ones who contain one long taking activity.

If the software task can be broken into smaller repeating tasks, then we can redefine the use cases to release area locks after executing small part of the thread. Then the thread will acquire Area locks again. This way, we will give priority to the other waiting users; and consequently increase the responsiveness.

If we cannot break the software task, then it is better to replicate the information the use case needs and work on the copied data. For instance, usually the optimizers like LP solvers cannot be interrupted and calculation may take long time. In such a case, we propose to create the input for the solver and release locks before solver start execution. After the solver is finished, the thread acquires locks again.

7.7.Software task aborts

There are several situations where software tasks can be aborted after it started: (i) a transaction can be aborted by the user, (ii) a deadlock may occur and transaction may be automatically terminated, (iii) a system crash may occur. The architecture which we propose doesn't allow deadlock situations. However, a system crash or deadlock situation may occur because of implementation bugs, or other external problems. In such situations, it is important for a system to recover and perform inverse operations (Weikum, 1991). Inverse operations are the operations which return the system to the state before triggering the software task.

Determination of an inverse operation is a complicated process as it requires more than knowledge about the object which was triggered and the operation which was performed on that object. For example, deleted objects will no longer exist. Therefore it will be difficult to perform the inverse operation unless the information about deleted object is stored somewhere. Inverse operations may also depend on the state at which the primary operation was triggered (Weikum, 1991).

Hadzilacos et al. (1991) proposed a method for UNDO/REDO actions for RTDBS. They proposed to copy data of atomic units before transaction is started and keep it until transaction is committed. If a transaction is aborted before commit, then the system restores the initial state of the system.

The restore procedure also requires concurrency control as it will change the state of the system. The software tasks of LHO scheduling system are very short and have very small impact on the system. In addition, in a well-tested and approved system failures will occur very seldom. Therefore implementing recovery procedures for such software tasks may be also a burden on the system. Although, we don't have data on what would have happened if

we had recovery processes in place, we estimate that adding recovery processes will slower the scheduling system, thus it will reduce the responsiveness. Therefore we leave implementation of recovery procedures for future studies.

However, we consider user abortions within the scope of this study as they may occur frequently. In our architecture, we allow user to abort the software task in all input points. For that reason, we propose to only change the state of the system after all user inputs are acquired. For cases where user inputs are acquired after preliminary calculations and state changes, we propose to perform calculations and state changes on temporary fields/objects and do the permanent state changes after all inputs are acquired. For example, a typical LHO use case requires inputs when it is triggered and during validation of rules. The trigger generally requires the user to input various inputs like employee, time frame, etc. During rule checks, user may be asked to approve/disapprove violation of rule/constraint. For such cases we propose to do the permanent state changes after all rules are validated or approved by the planner.

8. Solution implementation

In this chapter, the proposed architecture will be implemented to allow for an analysis of its responsiveness. The analysis of responsiveness was based on two measures: (i) the run time of a variety of use cases (ii) the user perception of the software. The run time of use cases is defined according to the time elapsed between trigger and completion of a use case. A run time is a numerical measurement of the response time that can be used to judge the responsiveness of the system. The user perception is a subjective measurement that rates the overall system performance. The amount of possible use case combinations makes it infeasible to measure the responsiveness of all use cases by numerical values. Furthermore, the scheduling system can have many states. Therefore we have introduced the indicator 'user perception' as a measurement of the overall responsiveness of the system.

We implemented the proposed architecture in ICRON software for a large European homecare organization (LEHO). ICRON is a rule-based, object-oriented, finite-capacity planning and scheduling solution. ICRON is supplied by Icron Technologies. Icron Technologies is a global firm with expertise in industrial and operational processes, specialized in manufacturing and supply chain management solutions. (See http://www.icrontech.com/ for more company and product details.)

The scheduling problem of LEHO was decomposed into several separations. Responsiveness measurements were performed on the largest separation (in terms of employees and planners).

In order to measure the responsiveness of the implemented architecture, we defined three types of scenarios. We based the scenarios on the number of planners. The largest separation of LEHO has 20 planners. The first scenario was called 'regular planning day' scenario (RDPS). It was based on the fact that on a regular day, there are 5 ST planners and 15 LT planners working on the system. The second scenario was named 'busy day scenario' (BDS). This scenario was designed to test the system performance with simultaneous and interdependent planners. The third scenario was called 'global lock scenario' (GLS). It was designed to measure system responsiveness when global lock is used.

The details of the scenarios are as follows:

1. **Regular planning day scenario (RDPS)**: In this scenario there are 20 planners working on the system; 5 ST planners and 15 LT planners. We simulate a regular planning day where LT planners are extending the existing planning for one more week and ST planners are maintaining the existing planning. This scenario was tested for 1 day (8 hours).
2. **Busy day scenario (BDS)**: In BDS scenario, we simulate the worst case scenario in which for each Area there are two planners working simultaneously. In this scenario we have 10 ST planners and 10 LT planners. The scenario was tested for 4 hours.
3. **Global lock scenario (GLS)**: In GLS scenario we test the impacts of the global locks. We divided the planners into 2 groups. The task of 10 of the planners was to trigger use cases where global lock is used extensively (E.g. Share capacity with other Areas). The second 10 planners were planning normal tasks. This scenario was tested for 2 hours.

We tested each scenario in two parts. **Firstly**, for each of the scenarios, we measured the results of three use cases: Open Planning GANNT Charts, UC01 - Schedule operation, UC 04 - Plan holiday / absence. We asked the planners to trigger the tested use case at exactly the

same time and to measure the response time. The response time is the time between the trigger time of a use case and the time the use case is fully performed. The last action of each use case is to refresh the views of the user. The testers measured the time of each use case by holding a stop-watch in one hand and a mouse on the other. Whenever they trigger a use case, they start the stop-watch and they stop it when they see the result of their request (use case) on the screen. For each use case, we measured 8 runs per user. The details of the measured results can be seen in Appendix 4.

Secondly, we asked each user to rate the system performance at the end of each day. We defined an ordinal scale according to {Very good, Good, Acceptable, Bad, Very bad}. We asked every user individually to write down their response to the following question: "How would you rate the performance of the system on the following scale: {Very good, Good, Acceptable, Bad, Very bad}?" We also explained to the user that we will consider "Acceptable" as the minimum "PASS" grade of the system. In this case, a "PASS" refers to the system implementation in terms of its performance. As a second step of the performance rating, we gathered the users who were not satisfied with performance in one room and asked them the following questions in open form: *(i) Do you think that the system GUI and use cases are sufficient to fulfil your expectation? (ii) Were you able to find the information you needed quickly? (iii) Were you able to execute planning actions fast and accurately? (iv) What action(s) (use cases) do you consider as not adequate? (v) Do you consider the system to perform inadequately during the entire test?* With the first two questions we wanted to identify the root cause of the users' dissatisfaction. It may well be possible for a user to be dissatisfied because she can't read/understand the GUI, or because he find the GUI not to be user-friendly or because the implemented use cases are not meeting her needs in terms of the process. For example, a user may think that the system is slow because it requires a lot of inputs, some of which could be gathered automatically. The other 3 questions were designed to discover which use cases are not performing well under what circumstances.

Table 6 displays the results of our tests. The norm values are defined by the LEHO organization. The average value for each scenario is calculated by taking the average of the average times of each user. The maximum value for each scenario is calculated by taking the maximum of all runs. User rating is calculated by taking the averages of each user. In order to take the average, we converted the scale {Very good, Good, Acceptable, Bad, Very bad} into numeric scale {5, 4, 3, 2, 1}.

Table 6: Test results

	"Open Planning GANTT charts"		UC 01		UC 04		User rating*
	Avg (sec)	Max (sec)	Avg (sec)	Max (sec)	Avg (sec)	Max (sec)	
Norm Values (Time cons.)	**2**	**4**	**2**	**3**	**11**	**12**	**Acceptable**
RDPS	1.8	4.1	1.5	2.5	3.2	7.2	Good
BDS	2.2	4.1	2.3	5	4.6	9.9	Acceptable
GLS	2.3	5.2	2.4	4.6	4.2	8.4	Acceptable

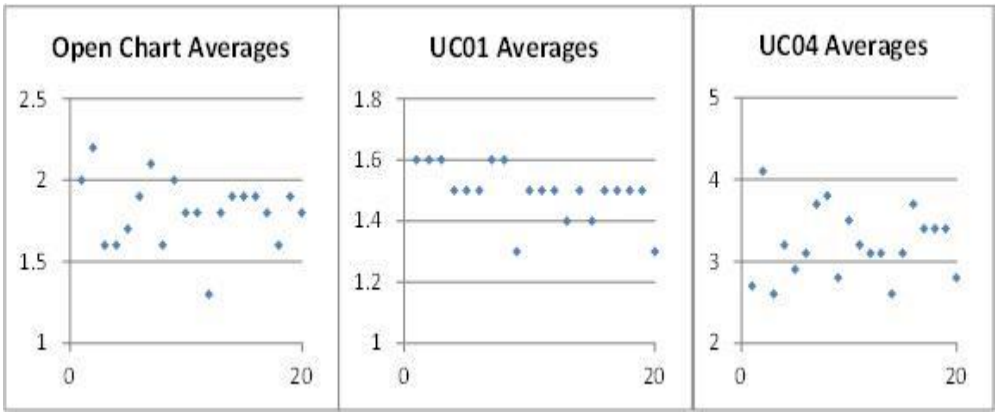
*Scale: Very good, Good, Acceptable, Bad, Very bad

Our findings from the tests are as follows:

A. **RPDS:** In the regular scenario, the results were interpreted as "Good" by the planners. The measured results were below the Norm response time for most of the cases. Figure 18 displays the average response time of each user and Figure

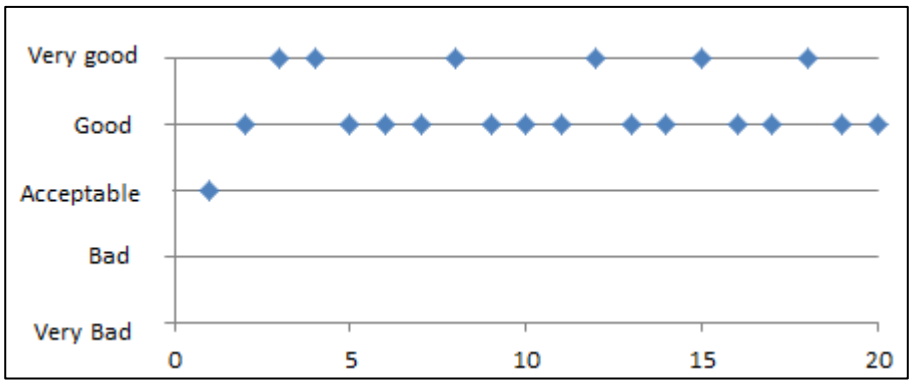
19 displays the user ratings. We didn't perform the second part of the user ratings questionnaire in this scenario because all of the ratings were "Acceptable" or above.

Figure 18: Average response times for RPDS*



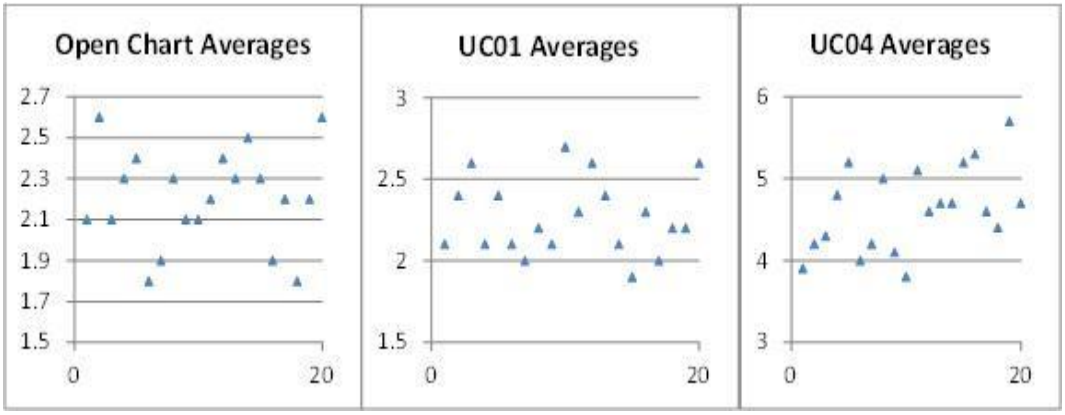
*Each point represents an average response time of a user

Figure 19: User ratings for RPDS



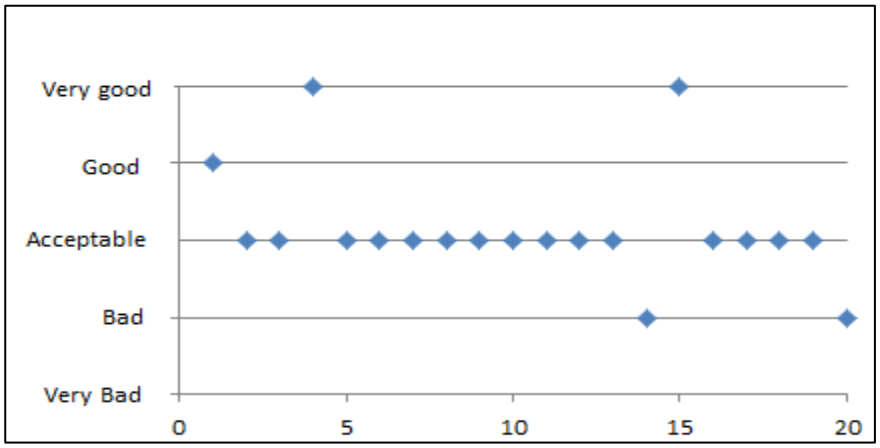
B. **BDS**: After the BDS test, the planners interpreted the responsiveness of the system as "acceptable" but with the remark that the switching between views must be improved. In the BDS scenario, If one of the planners is triggering the use cases very fast, the second planner will have to wait for the refresh actions at her screen. In such cases, switching between views is not very responsive. The measured results of the use cases indicate that the timing constraints (Norm values) are violated only slightly for some of the use cases. Figure 20 displays the average response time of each user and Figure 21 displays the user ratings. Based on the results displayed in Figure 21, we interviewed the two planners who rated the system performance as bad. During our interview, we found out that both users had more than 7 reports/charts open simultaneously. Therefore they had to wait long time for their client to refresh the open reports/charts after each use case run. As a solution to such cases, we propose two solutions: (i) to create workspaces for planners so that they can open only the views that they need during the use process they perform. For example, a planner requires different views for planning tasks than planning summer holidays, (ii) to refresh only the part of the views which are changed. The implementation of the second approach is more related to GUI architecture than the scheduling system architecture.

Figure 20: Average response times for BDS*



*Each point represents an average response time of a user

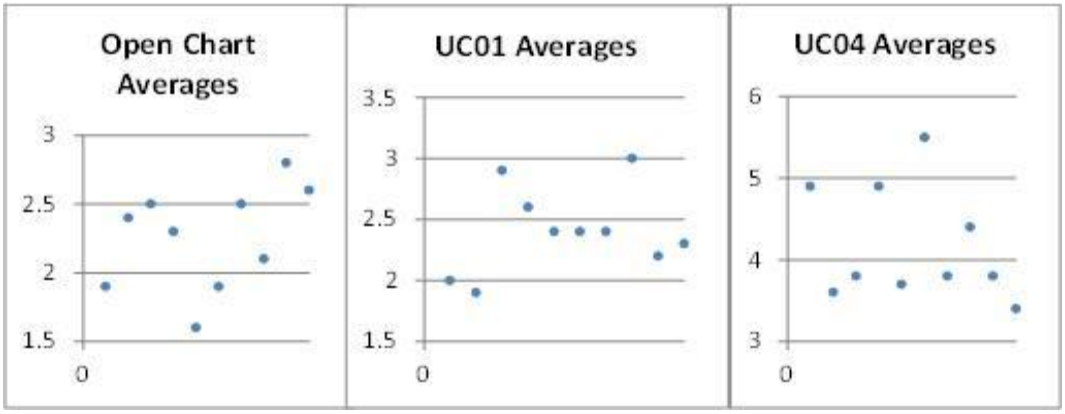
Figure 21: User ratings for BDS



C. **GLS**: After the GLS test, the planners interpreted the responsiveness of the system as "Acceptable". The users added a remark that there were fluctuation between the response times but in general the performance was interpreted as "Acceptable". The measured results also indicate that timing constraints (Norm values) are not violated most of the times but there is considerable fluctuation in the response times. Figure 22 displays the average response time of each user and Figure 23 displays the user ratings. Based on the results displayed in Figure 23, we interviewed the three planners who rated the system performance as bad. During our interview, we spotted two potential problems:

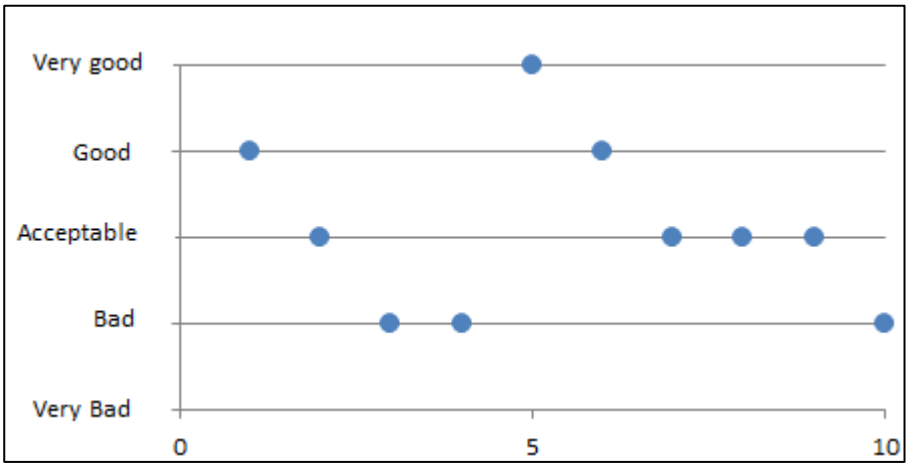
1. Similar to the problem with the BDS scenario, one of the users was working with minimum 8 charts/reports at the same time.
2. Two of the users were from one Area. We identified that this Area had a double number of employees/clients compared to the regular area. This indicates that GUI actions are slower for larger Areas. As a solution we propose to reduce the size of the GUI report/charts as not all of the information is required at the same time.

Figure 22: Average response times for GLS[+]



[+]Each point represents an average response time of a user

Figure 23: User ratings for GLS



9. Conclusion

The Scheduling of a large homecare organization is a complicated task because of the following three problem characteristics (i) the problem size is very big (in terms of the amount of staff and clients); (ii) the existence of numerous fuzzy objectives and soft constraints; (iii) a highly uncertain environment.

Achieving responsiveness for Large Homecare Organizations is a very complex task as it has to deal with distributed, co-dependent decision making. LHO, as well as the decision makers (planners) themselves, are distributed across wide geographical areas, where the decision of one planner influences the decision making of the other.

In this research, a software architecture was constructed in order to help LHO achieve a responsive scheduling process. This architecture was defined in four steps. Firstly, a proposal was made to separate the schedule into independent sub schedules. It was recognized that in homecare environments, the shared resources and organizational policies are the most influential factors determining separation. Secondly, a scheduling system object model was proposed, that is suitable for the handling of low-level scheduling processes in a distributed, multi-user environment. Thirdly, a centralized scheduling system was developed that is capable of handling shared resources and high level organizational policies. In order to protect the consistency of the centralized schedule, we proposed to use data locks. As an atomic locking unit, we propose the use of Area (or Team) so as to reduce the maintenance requirements of the software and to facilitate a responsive scheduling process. As a last step we proposed to break the software tasks that take a long time into subparts. This will allow other software task to be processed without waiting too long in a queue.

The schedule separation method that we propose is a pessimistic approach that separates the schedule into identical subparts. During the implementation of the architecture in a pilot organization we identified possibilities to separate the schedule into subparts of different size. For example, resource sharing is possible for the cities which are close to each other. If there are two cities that are close to each other, the separation method will propose to define a scheduling system for a unit that is larger than a city. However, in reality there are only a few cities that are close to each other. Therefore, the scheduling problem can be separated into non-identical subparts. Reducing the size of the separations will improve the overall system responsiveness. However, it will increase the complexity in managing the software. For example, all processes (scheduling input data maintenance) must be designed to support different separations.

We aimed to design an object model that is suitable for the handling of low-level scheduling processes. However, an object model for a scheduling system has several other components such as interfacing to external systems, schedule optimization, etc. A system designer must enhance the scheduling system architecture by adding these components to the design. For example, a scheduling system is almost unacceptable if it does not have interface to backbone systems or if it does not support outgoing interfaces like email or SMS.

We propose to use Area (or Team) as the atomic locking unit because the scheduling system is responsive and the maintenance of the software is simpler. However, during our research we also tested two other atomic locking units. The first scenario we tested allowed only one edit action at a time, i.e. used global lock. The run times of use cases are relatively low (1 to 5 seconds). Therefore, it may be possible for users to work without interfering with each other. Using global locks leads to acceptable results for five users. However, when we increased the number of users to 20, we observed run times which are five times higher than using Area locks. The second scenario was to use employee and client as atomic locking unit.

However, this brings complexity in management of locks as the locks needs to be recalculated based on the triggered use case. Moreover, for the same use case, different locks may need to be obtained over time. For instance, employee contracts are changing over time. Therefore when high level counters are calculated, then the locks to be acquired (see section 6 for definition) need to be recalculated. In some cases, recalculation of locks may take more time than the use case itself. Therefore, using employee or client locks does not always result in a responsive system. Additionally, using thousands of locks increases the chance of falling into deadlock. Resolving a deadlock requires tremendous amount of effort.

The results of implementation of our architecture in two large homecare organizations showed that the proposed architecture is responsive in most of the scenarios. In one of the pilot organizations, we observed that long term planners can extend the planning of one Area for one week in 20% of the time she needed before. Previously an organization was using spread sheets as a planning tool.

Our solution focuses on an architecture which handles existing scheduling processes. For future research, it might be beneficial to study process change possibilities that could simplify the schedule separation.

Additionally, it may be beneficial to test the architecture with client-based, periodical refreshes, where each client refreshes itself periodically instead of the server refreshing it automatically after each change.

10. Reflection

The thesis aimed to provide guidelines for homecare software architecture developers. The focus of the proposed software architecture is responsiveness in a multi-user environment where software users are located at different locations. Although the software will serve as a basic architecture for scheduling systems designed to serve large homecare organizations, not all components of a complete scheduling software program are included. To identify the scope and limitations of the current findings, the dimensions by which the literature was reviewed will be applied to the current study.

In the Literature Review chapter we have categorized the relevant literature in terms of the requirements of the current study. We will use the same categorization to rate the architecture that we propose. The analysis of the requirements applied to the current study are as follows:

- **Decentralized** [D]: The architecture that we propose supports decentralized decision making where multiple decision makers are collaborating in order to obtain a schedule for the organization. In order to support distributed, co-dependent decision making we propose a suitable object model and we propose a locking method to handle dependent decision making.
- **Multi-User Support** [MUS]: Our architecture enables multiple users to work simultaneously on the same schedule. However, it doesn't define the procedures for handling parallel threads and different queuing models.
- **Soft Timing Constraints** [STC]: We measure the responsiveness of our architecture by inserting soft timing constraints to use cases and we obtain acceptable results for the implementation we do for LEHO. However, we think that it needs to be tested further with more use cases and preferably different organizational structures.
- **Local Decision Making** [LDM]: Most of the times, the planning decisions are local. Area level of locking enables the planner to work independently.
- **Global Policies** [GP]: The architecture which we propose supports common global polices.
- **Home/Health Care** [HC]: the object model and locking mechanism which we propose is suitable for team based organizational structures which work on task basis rather than shift basis.
- **Low-Level Process Management** [LLPM]: the architecture which we propose support low-level scheduling processes.

In addition to the aforementioned requirements, a complete architecture must consider many other aspects of the scheduling system, such as optimization modules, interfaces and additional and custom constraints. The most important aspects requiring further consideration are as follows:

Organizational structure: The software architecture was designed for large homecare organizations that deploy a team based organizational structure. The proposed architecture may not provide as good a performance in an organization with a different organizational structure than the team based structure. For example, there are organizations that expect the employees to manage their own schedule. Every employee then serves a specific number of clients. The assignment of employees to clients doesn't change unless something critical happens. In such situations, Area locking may be unnecessary. Wider research about the organizational structures of homecare companies may be performed to determine the common organizational structures.

Transaction management: In our research, we do not consider the transaction management of deadlock or system crashes as software tasks of LHO scheduling system are very short and have very small impact on the system. In addition, in a well-tested and approved system, failures will occur very seldomly. Therefore, implementing recovery procedures for reversing the executed transactions may also be a burden on the system. However, we think that it may be useful to research transaction management methods for scheduling use cases which have big impact on the system state. For example, the impact of a use case which proposes a schedule for a city for two weeks will have a large effect on the system. Therefore a transaction management method needs to be considered by the software designers.

Other locks: In our study, we propose to use "Area" level locks in order to protect the feasibility of the plan. One of the alternatives we considered before choosing "Area" level locks was to use Employee\Client level locks where schedules are obtained by assigning a task of a client to an employee. We chose for an "Area" lock because it allows for the separation of lock management from use case definitions as it simplifies the development and maintenance of the software. However, Employee\Client level locks may result in a more responsive system. Therefore, for future research, we suggest to investigate possibilities for implementing Employee\Client locks in LHO scheduling systems.

Integration with other systems: In our research, we do not consider integration with other systems as part of the scheduling system architecture because the focus of our study is on responsiveness. However, as the scheduling system requires inputs from various sources (e.g. ERP systems, personnel system, CRM, backbone systems, payroll systems, etc.) and requires the provision of various outputs (e.g. email, personal agenda, schedule reports, SMS, etc.) we advise the system architects to consider integration with other systems when designing their architecture.

Physical infrastructure: Our research is centred on the responsiveness of the LHO scheduling system. However, the scheduling system responsiveness also depends on the physical resources that the software utilizes. For example, the network speed has significant effect on responsiveness as a relatively large amount of data needs to be transferred from server to the planner's location. Therefore we advise the system architects to consider the physical IT infrastructure in their architecture.

Graphical User Interface (GUI): During our implementation tests, we discovered that Graphical User Interface actions like refresh has significant effect on responsiveness as the duration of refresh is relatively high compared to the execution time of the use cases like "Schedule operation". Therefore we suggest system architects to consider system performance when designing GUI interactions.

11. Bibliography

Abbott, R. K., & Garcia-Molina, H. (1992). Scheduling real-time transactions: a performance evaluation. ACM Trans. Database Syst., 17(3), 513-560.

Akjiratikarl, C., Yenradee, P., & Drake, P. R. (2007). PSO-based algorithm for home care worker scheduling in the UK. Computers & Industrial Engineering, 53(4), 559-583.

Baccouche, L. (2005). Scheduling Multi-Class Real-Time Transactions: A Performance Evaluation. World Academy of Science, Engineering and Technology, 11, 65-68.

Baris, E. (2008). Home care in Europe: World Health Organization.

Begur, S. V., Miller, D. M., & Weaver, J. R. (1997). An Integrated Spatial DSS for Scheduling and Routing Home-Health-Care Nurses. Interfaces, 27(4), 35-48.

Ben-Shaul, I. Z., Kaiser, G. E., & Heineman, G. T. (1992). An architecture for multi-user software development environments. Paper presented at the Proceedings of the fifth ACM SIGSOFT symposium on Software development environments.

Berrada, I., Ferland, J. A., & Michelon, P. (1996). A multi-objective approach to nurse scheduling with both hard and soft constraints. Socio-Economic Planning Sciences, 30(3), 183-193.

Berrada, I., Ferland, J. A., & Michelon, P. (1996). A multi-objective approach to nurse scheduling with both hard and soft constraints. Socio-Economic Planning Sciences, 30(3), 183-193.

Borsani, V., Matta, A., Beschi, G., & Sommaruga, F. (2006, 26 February 2007). A Home Care Scheduling Model For Human Resources. Paper presented at the Service Systems and Service Management, Troyes.

Bosman, R., Bours, G. J. J. W., Engels, J., & de Witte, L. P. (2008). Client-centred care perceived by clients of two Dutch homecare agencies: A questionnaire survey. International Journal of Nursing Studies, 45(4), 518-525.

Bricon-Souf, N., Anceaux, F., Bennani, N., Dufresne, E., & Watbled, L. (2005). A distributed coordination platform for home care: analysis, framework and prototype. International Journal of Medical Informatics, 74(10), 809-825.

Carrascosa, C., Bajo, J., Julian, V., Corchado, J. M., & Botti, V. (2008). Hybrid multi-agent architecture as a real-time problem-solving model. Expert Systems with Applications, 34(1), 2-17.

Corchado, J. M., Bajo, J., de Paz, Y., & Tapia, D. I. (2008). Intelligent environment for monitoring Alzheimer patients, agent technology for health care. Decision Support Systems, 44(2), 382-396.

Date, H., & Matsuo, T. (2008). Effects of At-Home Nursing Service Scheduling in Multiagent Systems. In N. Nguyen & R. Katarzyniak (Eds.), New Challenges in Applied Intelligence Technologies (Vol. 134, pp. 245-254-254): Springer Berlin / Heidelberg.

Ernst, A. T., Jiang, H., Krishnamoorthy, M., & Sier, D. (2004). Staff scheduling and rostering: A review of applications, methods and models. European Journal of Operational Research, 153(1), 3-27.

Eveborn, P., Flisberg, P., & Rönnqvist, M. (2006). Laps Care--an operational system for staff planning of home care. European Journal of Operational Research, 171(3), 962-976.

Fraile, J., Bajo, J., Lancho, B., & Sanz, E. (2009). HoCa Home Care Multi-agent Architecture International Symposium on Distributed Computing and Artificial Intelligence 2008 (DCAI 2008) (pp. 52-61).

Frankovič, B., Budinská, I., & Tung, D. T. (2002). Creation of Ontology for Planning and Scheduling. Paper presented at the 3 International Symposium on Computational Intelligence.

Graham, J. R. (2001). Real-time scheduling in distributed multi agent systems. University of Delaware.

Guo, Q.-l., & Zhang, M. (2009). Multiagent-based scheduling optimization for Intelligent Manufacturing System. The International Journal of Advanced Manufacturing Technology, 44(5), 595-605-605.

Hadzilacos, T., & Hadzilacos, V. (1991). Transaction synchronisation in object bases. Journal of Computer and System Sciences, 43(1), 2-24.

Haritsa, J. R., Livny, M., & Carey, M. J. (1991, 4-6 Dec 1991). Earliest deadline scheduling for real-time database systems. Paper presented at the Real-Time Systems Symposium, 1991. Proceedings., Twelfth.

Huang, J., Jennings, N. R. and Fox, J. (1994). Cooperation in Distributed Medical Care. 2nd Int. Conf. on Cooperative Information Systems (CoopIS-94), Toronto, Canada.

Huang, J., Jennings, N. R. and Fox, J. (1995). An Agent-based Approach to Health Care Management. Int. Journal of Applied Artificial Intelligence, 9(4), 401-420.

Hutzschenreuter, A. K., Bosman, P. A. N., Blonk-Altena, I., van Aarle, J., & La Poutre, H. (2008). Agent-based patient admission scheduling in hospitals. Paper presented at the 7th international joint conference on Autonomous agents and multiagent systems, Estoril, Portugal.

Isern, D., Sanchez, D., & Moreno, A. (2010). Agents applied in health care: A review. International journal of medical informatics, 79(3), 145-166.

Israel, Z. B.-S., & George, T. H. (1996). A three-level atomicity model for decentralized workflow management systems. Distributed Systems Engineering 3(4), 239.

Itabashi, G., Chiba, M., Takahashi, K., & Kato, Y. (2005). A Support System for Home Care Service Based on Multi-agent System. Paper presented at the Information, Communications and Signal Processing, 2005 Fifth International Conference on.

Julian, V., & Botti, V. (2004). Developing real-time multi-agent systems. Integrated Computer-Aided Engineering, 11(2), 135-149.

Koutkias VG, Chouvarda I, & N., M. (2005). A multiagent system enhancing home-care health services for chronic disease management. IEEE Trans Inf Technol Biomed, 9(4), 528-537.

Kruchten, P. (2004). The Rational Unified Process and Introduction (3 ed.).

Li, Y., Huang, B. Q., Liu, W. H., Gou, H. M., & Wu, C. (2002). Ontology and multi-agent based decision support for enterprise bidding. 2001 Ieee International Conference on Systems, Man, and Cybernetics, Vols 1-5, 2947-2951

3494.

Liu, W., Flood, I., & Issa, R. R. A. (2006). A structured micro-level planning and scheduling method. Proceedings of 2006 International Conference on Construction & Real Estate Management, Vols 1 and 2, 1-5

1692.

Liu, Z., Zheng, H. R., Yan, R. Y., & Gao, J. (2001). Studies on agent-based cooperation for virtual enterprises. Computer Science and Technology in New Century, 285-289

630.

Mackworth, A. K., & Zhang, Y. (2003). A Formal Approach to Agent Design: An Overview of Constraint-Based Agents. Constraints, 8(3), 229-242.

Merdan, M., Koppensteiner, G., Zoitl, A., & Favre-Bulle, B. (2007). Distributed Agents Architecture Applied in Assembly Domain. Paper presented at the International Symposium on Knowledge and Systems Sciences.

Moreno, A., Valls, and Riano, D (2004). Medical applications of multi-agent systems. presented at ECAI Workshop on Agents Applied in Health Care, Valencia, Spain.

Motta, E., Rajpathak, D., Zdrahal, Z., & Roy, R. (2002). The Epistemology of Scheduling Problems. Paper presented at the Os the 15 European Conference on Artificial Intelligence.

Nealon, J., & Moreno, A. (2003). Agent-Based Applications in Health Care: Verlag.

Neelamkavil, J., Diaz, A., & Graefe, U. (1992). Investigation of rule-based, object-oriented and blackboard systems for scheduling. Paper presented at the Flexible Automation 1992.

Pinelle, D., & Gutwin, C. (2001). Collaboration Requirements for Home Care. Research Report, University of Saskatchewan.

Pinelle, D. a. G., C. (2001, 2003). Awareness-Based Scheduling in a Home Care Clinical Information System. Paper presented at the AMIA Annu Symp Proc.

Rajpathak, D., Motta, E., & Rajkumar, R. (2001). A Generic Task Ontology for Scheduling Applications. Paper presented at the Artificial Intelligence (IC-AI'2001).

Ramamritham, K. (1993). Real-time databases. Distributed and Parallel Databases, 1(2), 199-226-226.

Begur, S. D. M. a. J. W. (1997). An integrated spatial DSS for scheduling and routing home-health-care nurses. Interfaces, 27(4), 35-48.

Santos, R., Lipari, G., & Santos, J. (2008). Improving the schedulability of soft real-time open dynamic systems: The inheritor is actually a debtor. Journal of Systems and Software, 81(7), 1093-1104.

Ulusoy, Ö., & Belford, G. G. (1993). Real-time transaction scheduling in database systems. Information Systems, 18(8), 559-580.

Wang, L. (2010). A Novel Collaborative Planning Approach for Digital Manufacturing. In G. Huang, K. Mak & P. Maropoulos (Eds.), Proceedings of the 6th CIRP-Sponsored International Conference on Digital Enterprise Technology (Vol. 66, pp. 939-955-955): Springer Berlin / Heidelberg.

Weikum, G. (1991). Principles and realization strategies of multilevel transaction management. ACM Trans. Database Syst., 16(1), 132-180.

Weikum, G., & Hasse, C. (1993). Multi-level transaction management for complex objects: implementation, performance, parallelism. The VLDB Journal, 2(4), 407-454.

Weikum, G., & Schek, H.-J. (1992). Concepts and applications of multilevel transactions and open nested transactions Database transaction models for advanced applications (pp. 515-553): Morgan Kaufmann Publishers Inc.

12. Appendix 1: Use cases

This chapter will define the use cases that are relevant for this case. The use cases defined below will be used for both operational planning and dispatching.

Use Case 1: Schedule operation

Primary Actor:	Planner
Goal in context:	Planner assigns a task to an employee
Scope:	Scheduling system
Level:	Summary
Stakeholders and Interests:	A. Planner wants to assigns a task to an employee and check validity of the assignment as fast as possible. B. Planner wants to update statistics and counters like remaining capacity, remaining operations to be planned, etc. automatically. C. Related planners want to be informed about the changes. D. Management of the company wants to satisfy the demand with minimum resource usage. E. Government wants to spend as less as possible but also satisfy the clients.(mainly governments are paying for the services)
Precondition:	I. The planner is logged to the scheduling system. II. There is task to be planned. III. There is employee to be planned. IV. The planner is authorized to assign the task to the employee.
Minimal guarantee:	Assignment of a task to an employee takes less than 2 seconds
Success guarantee:	The task is assigned to the employee. Statistics and counters are updated accordingly. Transaction is recorded to the database. All related planner's views are refreshed automatically.

Main success scenario:	1. The planner sets the inputs: the employee who will perform the task, the task to be planned, start time of the task and end time of the task. 2. The system assigns the task to the employee at requested start and end time. 3. The system recalculates the statistics and counters (See “Appendix 2: Statistics and Counters” for details). 4. The system notifies the related planners about the change.
Extensions:	1a. The planner enters invalid inputs(employee or client cannot be found, start and end time are not applicable for the selected task) 1a1. System notifies the user and requests the inputs again. 2a. The system fails to validate one or more hard constraints. 2a1. The system notifies the planner about the violation and returns to step 1 2b. The system fails to validate one or more soft constraints and planner accepts the violation. 2b1. The system continues with step 3. 2c. The system fails to validate one or more soft constraints and planner rejects the violation. 2c1. The system terminates the use case.

Use Case 2: Un-schedule operation

Primary Actor:	Planner
Goal in context:	Planner removes a previously assigned task from an employee
Scope:	Scheduling system
Level:	Summary
Stakeholders and Interests:	A. Planner wants to remove previously assigned task from the employee. B. Planner wants to update statistics and counters like remaining capacity, remaining operations to be planned, etc. automatically. C. Related planners want to be informed about the changes. D. Management of the company wants to satisfy the demand with minimum resource usage. E. Government wants to spend as less as possible but also satisfy the clients.(mainly governments are paying for the services)
Precondition:	V. The planner already has the scheduling system open. VI. There is task that is planned on the selected employee. VII. The planner is authorized to remove the task from the employee.
Minimal guarantee:	Removing of a task from an employee takes less than 2 seconds
Success guarantee:	The task is removed from the employee's tasks to be performed. Statistics and counters are updated accordingly. Transaction is recorded to the database. All related planners are notified about the change.
Main success scenario:	1. The planner selects the task to be removed. 2. The system removes the task from the employee's agenda. 3. The system recalculates the statistics and counters (See "Appendix 2: Statistics and Counters" for details). 4. The system notifies the related planners about the change.
Extensions:	

Use Case 3: Schedule all operations

Primary Actor:	Planner
Goal in context:	Planner assigns set of tasks to an employee
Scope:	Scheduling system
Level:	Summary
Stakeholders and Interests:	A. Planner wants to assigns a task to an employee and check validity of the assignment as fast as possible. B. Planner wants to update statistics and counters like remaining capacity, remaining operations to be planned, etc. automatically. C. Related planners want to be informed about the changes. D. Management of the company wants to satisfy the demand with minimum resource usage. E. Government wants to spend as less as possible but also satisfy the clients.(mainly governments are paying for the services)
Precondition:	I. The planner is logged to the scheduling system. II. There is task to be planned. III. There is employee to be planned. IV. The planner is authorized to assign the task to the employee.
Minimal guarantee:	-
Success guarantee:	As much as possible fixed tasks are planned. Statistics and counters are updated accordingly. Transaction is recorded to the database. All related planner's views are refreshed automatically.
Main success scenario:	1. The planner sets the inputs: choose the *Area* to be planned, start time of the period to be scheduled and end time of the period to be scheduled. 2. The system determines the most suitable(based on the rules defined by the organization) employee for each task and assigns the tasks to the employee. 3. The system recalculates the statistics and counters (See “Appendix 2: Statistics and Counters” for details). 4. The system notifies the planner with the number of planned

	tasks, not planned tasks and the reasons for not planned tasks. 5. The system notifies the related planners about the change.
Extensions:	1a. The planner enters invalid inputs(*Area* cannot be found, start and end time are not applicable) 1a1. System notifies the user and requests the inputs again.

Use Case 4: Plan holiday / absence

Primary Actor:	Planner
Goal in context:	Planner creates and approves the holiday request of an employee
Scope:	Scheduling system
Level:	Summary
Stakeholders and Interests:	A. Planner wants to create a holiday for an employee and check validity of the assignment as fast as possible. B. Planner wants to update statistics and counters like remaining capacity, violated operations, *Area* continuity, etc. automatically. C. Related planners want to be informed about the changes. D. Management of the company wants to have enough capacity to satisfy the demand.
Precondition:	I. The planner is logged to the scheduling system. II. There is employee is allowed to have a holiday. III. The planner is authorized to approve the holiday of the employee.
Minimal guarantee:	-
Success guarantee:	The holiday request is created and approved. Statistics and counters are updated accordingly. Transaction is recorded to the database. All related planner's views are refreshed automatically.
Main success scenario:	1. The planner sets the inputs: the employee, the holiday type to be requested, start time of the holiday and end time of the holiday. 2. The system creates the holiday request. 3. The system recalculates the statistics and counters (See “Appendix 2: Statistics and Counters” for details). 5. The system notifies the related planners about the change.

Extensions:	1a. The planner enters invalid inputs(*Area* cannot be found, start and end time are not applicable) 1a1. System notifies the user and requests the inputs again. 2a. The system fails to validate one or more hard constraints. 2a1. The system notifies the planner about the violation and returns to step 1 2b. The system fails to validate one or more soft constraints and planner accepts the violation. 2b1. The system continues with step 3. 2c. The system fails to validate one or more soft constraints and planner rejects the violation. 2c1. The system terminates the use case.

Use Case 5: Loan employee to another region(area)

Primary Actor:	Planner
Goal in context:	Planner of one *Area* loans an employee to another *Area*
Scope:	Scheduling system
Level:	Summary
Stakeholders and Interests:	A. Planner wants to create a holiday for an employee and check validity of the assignment as fast as possible. B. Planner wants to update statistics and counters like remaining capacity, violated operations, *Area* continuity, etc. automatically. C. The planner of the borrowing *Area* wants extra capacity to satisfy the demand on her Area. D. Related planners want to be informed about the changes. E. Management of the company wants to have enough capacity at each location to satisfy the demand.
Precondition:	I. The planner is logged to the scheduling system. II. There is employee has an empty space in her schedule.
Minimal guarantee:	-
Success guarantee:	The employee is loaned to the holiday request is created and approved. Statistics and counters are updated accordingly. Transaction is recorded to the database. All related planner's views are refreshed automatically.
Main success scenario:	1. The planner sets the inputs: the employee, the *Area* to be loaned, start time of the loaning period and end time of the loaning period. 2. The system creates the loaned capacity object. (See "Shared resource classes" section for details) 3. The system recalculates the available capacity for the two *Area*'s. 4. The system recalculates the statistics and counters (See "Appendix 2: Statistics and Counters" for details). 5. The system notifies the related planners about the change.

Extensions:	1a. The planner enters invalid inputs(*Area* cannot be found, start and end time are not applicable) 1a1. System notifies the user and requests the inputs again. 2a. The system fails to validate one or more hard constraints. 2a1. The system notifies the planner about the violation and returns to step 1 2b. The system fails to validate one or more soft constraints and planner accepts the violation. 2b1. The system continues with step 3. 2c. The system fails to validate one or more soft constraints and planner rejects the violation. 2c1. The system terminates the use case.

13. Appendix 2: Statistics and Counters

Statistics is the general name for aggregate values calculated based on the schedule outputs. The statistics that are relevant for this case are:

- **Capacity utilization**: per employee, per Area, Per city, per province
- **Continuity**: per Area, per city. It is calculated by dividing the total hours of absences and holidays by contracted hours of the employees.
- **Remaining hours**: Remaining available hours per employee, per Area
- Overtime hours: Per employee, per Area, per week and possible combinations

Counters are the values which are counted based on the scheduling process output. The counters that are relevant for this case are:

- **Number of visits**: The number of times an employee visited a client.
- **The number of violated weekly rests**: the number of times the weekly rest of the employee is violated.
- **The number of violated daily rests**: the number of times the daily rest of the employee is violated.
- **The number of violated characteristics:** the number of times the characteristic rules are for the employee/client
- **The number of planned task:** of client, to the employee

14. Appendix 3: Constraints

This section will first describe the main constraints in scheduling homecare tasks from the perspective of the employee, the client and the organization (LHO). An example schedule will be used to illustrate the main set of constraints that a planner must consider during the scheduling process. At the end of the chapter, we will summarize the scheduling difficulties of LHO.

An example schedule

Table 7 shows an imaginary schedule output for *Taner Chakar*. *Taner Chakar* is an employee of *Company A* at location *Area 1.*

The scheduling process of LHO is expected to provide an output which is similar to Table 7. The scheduling process matches the employees of the organization with the work that needs to be performed. The schedule of the LHO is obtained by assigning the *tasks* to the *employees*.

However, the assignment of tasks to employees is not a trivial process because each actor involved has different constraints. The following sections will describe the different constraints from the stakeholders' perspective.

Table 7: Example schedule for one employee Taner Chakar

Employee	Date	Client	Start	Finish	Duration (hrs.)	Priority	Location
Taner Chakar	26/04/2010	Client 1	08:00	12:00	04:00	Normal	Area 1
Taner Chakar	26/04/2010	Client 1	12:20	16:20	04:00	Normal	Area 1
Taner Chakar	27/04/2010	Client 1	12:20	16:20	04:00	Normal	Area 1
Taner Chakar	**28/04/2010**	Client 2	**08:00**	**12:00**	**04:00**	**Normal**	Area 1
Taner Chakar	**28/04/2010**	Client 3	**12:20**	**14:20**	**02:00**	**Normal**	Area 1
Taner Chakar	**29/04/2010**	Client 1	**14:20**	**16:20**	**02:00**	**Normal**	Area 1
Taner Chakar	06/05/2010	Client 4	12:20	16:20	04:00	Normal	Area 1
Taner Chakar	17/05/2010	Client 1	08:00	12:00	04:00	High	Area 1
Taner Chakar	17/05/2010	Client 1	12:20	16:20	04:00	Normal	Area 1
Taner Chakar	18/05/2010	Client 1	12:20	16:20	04:00	Normal	Area 1
Taner Chakar	20/05/2010	Client 1	08:00	12:00	04:00	Normal	Area 1
Taner Chakar	20/05/2010	Client 1	12:20	16:20	04:00	Normal	Area 1
Taner Chakar	25/05/2010	Client 1	08:00	12:00	04:00	Normal	Area 1
Taner Chakar	25/05/2010	Client 1	12:20	16:20	04:00	Normal	Area 1
Taner Chakar	26/05/2010	Client 1	12:20	16:20	04:00	Normal	Area 1

Schedule from the employee perspective

From an employee perspective, a schedule is not simply a list which contains the tasks to be performed. A schedule also contains meetings, holiday plans, absenteeism, etc. Figure 24 demonstrates the schedule of Taner Chakar in a GANNT form.

In addition to planned tasks, the GANTT chart of Taner Chakar also shows work patterns, holiday requests, absences and loaned periods (a period of time in which the employee works at different location than his contractor) of the employee.

When creating this schedule, a planner must consider large number of constraints of the employee like the number of contracted hours, preferences of employee, absenteeism, labour union rules, etc. For instance, when making the two day schedule of Taner Chakar, the planner considers the following scheduling constraints:

Employee Constraint 1 (EC1): An employee cannot be planned outside her work pattern.

In the example GANTT chart, Taner Chakar is only allowed to be planned at white zones.

Employee Constraint 2 (EC2): A task cannot overlap with a holiday.

Employee Constraint 3 (EC3): A task cannot overlap with an absence.

Employee Constraint 4 (EC4): A task cannot overlap with another task.

Employee Constraint 5 (EC5): An employee needs T_R hours of rest every day.

T_R differs for every country. A common number is 8.

Employee Constraint 6 (EC6): An employee needs continuous T_W hours break within every T_D days.

T_W andT_D differs for every country. Often T_W is set to 48 and T_D is set to 7.

Employee Constraint 7 (EC7): The total travel time within one day cannot be longer than T_T hours.

T_T differs for every country. A common number is 2.

Employee Constraint 8 (EC8): Employee's special conditions must be considered when assigning her to a task.

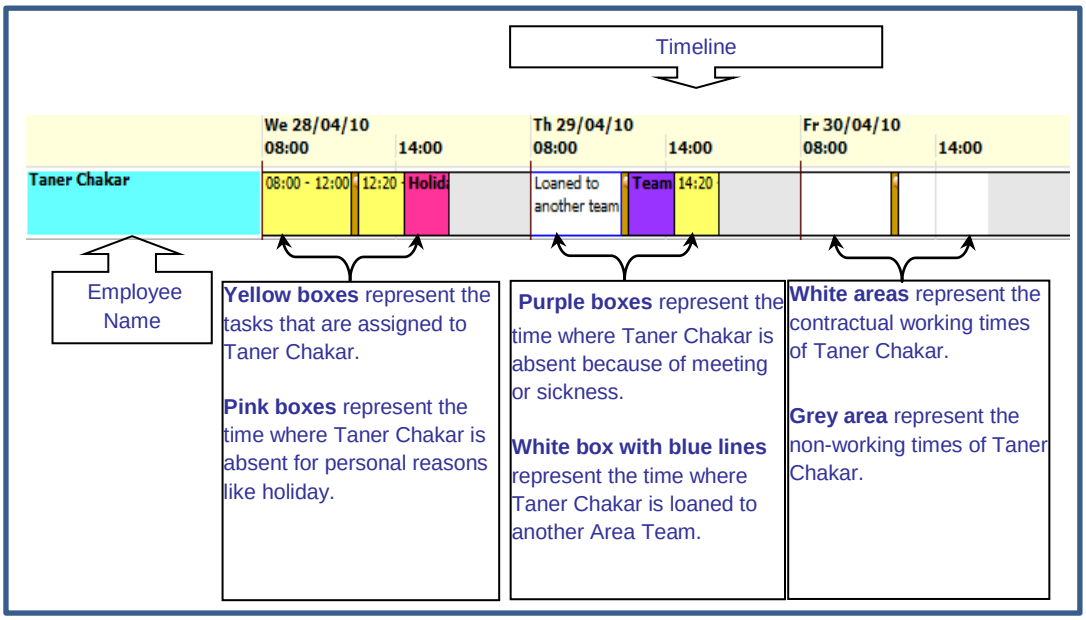
Employees can have special conditions which need to be considered before assigning her to tasks. For instance, an employee with allergy for pets shouldn't be assigned to a client with pet at home. Or a pregnant employee cannot be assigned to a task which involves heavy physical activities.

Employee Constraint 8: An employee's personal preference must be considered when assigning her to a task.

Employees have different type of preferences varying from the time they want to work to the clients they want to work, etc. Mostly, these preferences are soft constraints. It is very important that the planning takes them into account to increase employee satisfaction. For instance, consider an employee who doesn't want to work very early in the morning because her child leaves to school at 09:00(AM). Therefore, it would be disastrous to assign her a task which starts at 07:00(AM). In such a case the employee

has to make sure that her child gets the necessary support to wake up, go to school, etc. This could be a very hectic task which may reduce her energy to work.

Figure 24 GANTT chart for Taner Chakar



Schedule from the client perspective

Figure 25 demonstrates the schedule of Client 1, Client 2 and Client 3 in a GANNT form. The chart shows the same schedule as Figure 25 and Table 7 but from client perspective.

In addition to planned tasks, the client GANTT chart also shows the earliest start time and latest finish time of a task, and client absences.

When making the schedule given in Figure 25, a planner must consider the following constraints:

Client Constraint 1 (CC1): All tasks needs to be planned.

Client Constraint 2 (CC2): A task can only be planned between the earliest start and the latest finish time.

Client Constraint 3 (CC3): The tasks of a client mustn't overlap.

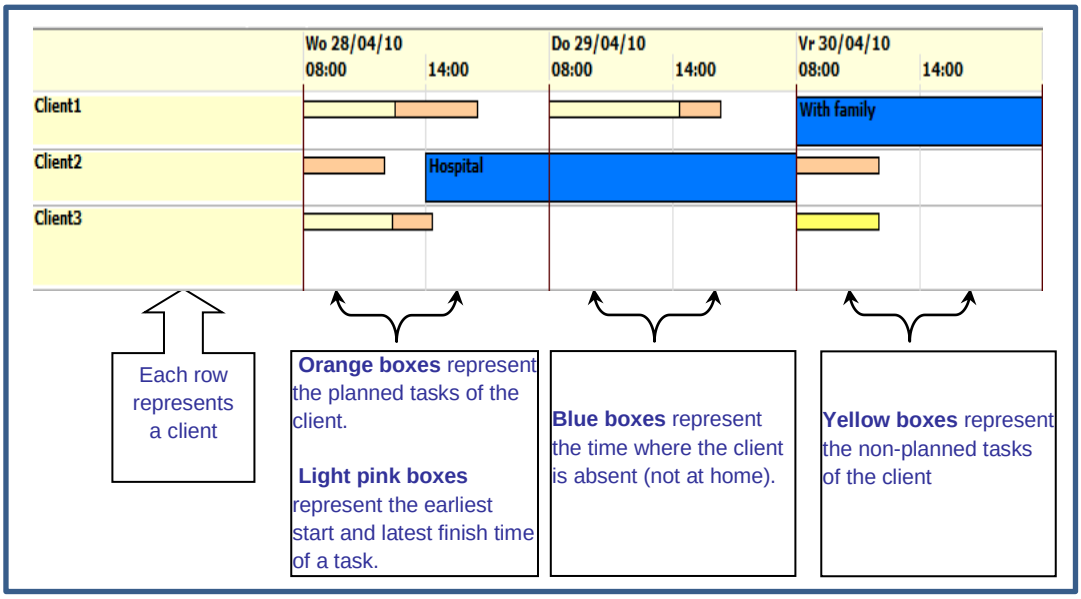
Client Constraint 4 (CC4): A task cannot overlap with client absence.

The client is not always at home. Sometimes they may go to the hospital or holiday. It is not desired to plan a task at a time where the client is not at her home.

Client Constraint 5 (CC5): Client's personal preference must be considered when assigning her tasks to employees.

The client may have personal preferences about the employee who delivers the service or about the time the help is delivered. For instance a client may prefer non-smoking employee.

Figure 25: GANTT chart for the Client 1, Client2 and Client 3



Schedule from the organizational/administration perspective

When making a planning, a planner must also consider organizational objectives, rules and policies. LHO need to consider the organizational constraints in order to increase customer satisfaction at minimal cost.

When making the planning represented in Table 7, the planner must consider the following constraints:

Organizational constraint 1: Tasks cannot be skipped.

All tasks needs to be planned to increase customer satisfaction and to obey governmental laws.

Organizational Constraint 2: An employee cannot be allowed to have more holidays than her contracted balance.

Organizational Constraint 3: An employee with low mobility can only be assigned to clients who are living in the same region.

The clients of the LHO are distributed over wide regions. It is not cost effective to assign an employee without a car to a client in a different city.

Organizational Constraint 4: The available capacity should not decrease below the minimum demand requirement.

Employees cannot be allowed to go on holiday in a period where the available capacity cannot satisfy the minimum demand requirement. The minimum demand requirement is determined in different ways. Some organizations define it as the sum of all urgent tasks; other organizations define it as the percentage of the total available capacity, etc.

15. Appendix 4: Test results

This appendix contains the measurements which were taken during the tests of LHO.

RPDS Scenario

User ratings

STP1	STP2	STP3	STP4	STP5	LTP1	LTP2	LTP3	LTP4	LTP5	LTP6	LTP7	LTP8	LTP9	LTP10	LTP11	LTP12	LTP13	LTP14	LTP15	Avg
Acceptable	Good	Very Good	Very Good	Good	Good	Good	Very Good	Good	Good	Good	Very Good	Good	Good	Very Good	Good	Good	Very Good	Good	Good	Good

Opening Charts

Run	STP1	STP2	STP3	STP4	STP5	LTP1	LTP2	LTP3	LTP4	LTP5	LTP6	LTP7	LTP8	LTP9	LTP10	LTP11	LTP12	LTP13	LTP14	LTP15	Avg	Max
1	2.2	2.7	1.4	1.2	1.7	2.8	1.6	1.5	1.4	1.3	1	1	1.6	1.2	1.6	1.4	1.9	1.8	1.5	1.4	1.6	2.8
2	1.5	1.7	1.5	1.3	1.9	1.6	2.2	1.1	2.2	2.4	2.2	1.6	1.9	1.6	2.5	1.6	1.4	1.3	1.9	2.3	1.8	2.5
3	4.1	2.4	2	1.3	1.1	1.9	1.7	1.9	3.1	2.5	1.3	1.3	1.5	2.7	1.9	1.1	1.7	2.6	3	1.1	2	4.1
4	1.4	1.7	3.2	1.2	1.3	1.3	1.9	1.1	1.9	2.5	1.4	1.3	2.9	1.8	3.1	2.2	2.6	1.2	1.3	1.7	1.9	3.2
5	2.5	2.5	1.3	2.2	2	1.3	1.6	1.6	2	1.2	2.4	0.9	1.1	1.8	1.3	2.8	1.5	1.3	1.8	1.8	1.7	2.8
6	1.5	1.8	1.1	1.6	1.9	1.8	2.3	1.5	1.5	1.3	1.7	1.3	1.7	1.5	1.9	1.6	1.1	2.1	2	2.4	1.7	2.4
7	1.2	1.8	1.1	2.5	1.8	2.5	2.9	1.3	2	1.5	2.2	1.1	1.5	1.8	1.4	2.5	2	0.9	1.2	1.1	1.7	2.9
8	1.6	3.2	1.1	1.4	1.9	1.9	2.2	2.9	2	1.9	2.5	2	2.5	3	1.4	1.9	1.9	1.6	2.7	2.5	2.1	3.2
Avg	2	2.2	1.6	1.6	1.7	1.9	2.1	1.6	2	1.8	1.8	1.3	1.8	1.9	1.9	1.9	1.8	1.6	1.9	1.8	1.8	
Max	4.1	3.2	3.2	2.5	2	2.8	2.9	2.9	3.1	2.5	2.5	2	2.9	3	3.1	2.8	2.6	2.6	3	2.5		4.1

UC 01

Run	STP1	STP2	STP3	STP4	STP5	LTP1	LTP2	LTP3	LTP4	LTP5	LTP6	LTP7	LTP8	LTP9	LTP10	LTP11	LTP12	LTP13	LTP14	LTP15	Avg	Max
1	2	1.2	1.1	1	1.6	0.9	2.1	1.7	1	1.6	1.7	1.3	1.3	1.5	1.8	1.3	1.7	1.8	1.2	1.4	1.5	2.1
2	1.7	1.7	2.3	1.3	1.2	1.1	1.5	1.7	1.4	1	1.2	1.6	1.6	1.6	1	2	1.8	1.6	1.7	1.2	1.5	2.3
3	1.4	1.8	1.7	1.7	1	1.8	2.2	2.2	1.5	2.3	1.3	1.5	1.1	2.5	1.2	1.5	1.6	1.6	1.6	1.4	1.6	2.5
4	1.3	1.6	1.4	1.4	1.7	2.2	1.4	1.5	1.2	1.5	1.7	2	1.2	1	1.9	1.6	1.3	1.5	1.5	1.6	1.5	2.2
5	1.6	1.3	1.3	1.4	2.3	1.3	1.3	1.1	1.2	1.3	1.2	1.6	1.2	1.5	1.3	1.1	1.3	1.2	1.1	1.2	1.3	2.3
6	1.6	1.5	1.8	1.5	1.4	1.8	1.2	2.1	1.5	2	1.9	1.3	1.3	1.6	1.3	1.8	1.7	1.3	1.3	1.5	1.6	2.1
7	1.8	2	1.4	2.3	0.8	1.3	1.1	1.5	1.2	1.3	1	1.6	1.6	1.5	1.4	1.2	0.8	1	1.6	1	1.4	2.3
8	1.6	1.7	1.4	1.3	2.2	1.3	1.6	1.3	1.4	1.1	1.9	1.3	1.9	1	1.4	1.1	2.1	1.9	1.6	1.4	1.5	2.2
Avg	1.6	1.6	1.6	1.5	1.5	1.5	1.6	1.6	1.3	1.5	1.5	1.5	1.4	1.5	1.4	1.5	1.5	1.5	1.5	1.3	1.5	
Max	2	2	2.3	2.3	2.3	2.2	2.2	2.2	1.5	2.3	1.9	2	1.9	2.5	1.9	2	2.1	1.9	1.7	1.6		2.5

UC 04

Run	STP1	STP2	STP3	STP4	STP5	LTP1	LTP2	LTP3	LTP4	LTP5	LTP6	LTP7	LTP8	LTP9	LTP10	LTP11	LTP12	LTP13	LTP14	LTP15	Avg	Max
1	4	2.8	3.1	1.9	5.6	4	3.3	4.4	2.9	3.3	2.6	3.2	3.7	3.2	2.6	3.9	7.1	3.4	3.1	3.1	3.6	7.1
2	2.2	7.2	2.7	3.3	2.5	2.9	3.9	4.1	3.7	3.2	3.2	3.1	2.2	2.3	2.3	2.9	2.9	2.3	2.4	5.1	3.2	7.2
3	1.8	4	2.7	2.8	2.7	4.6	4.8	4	2.6	4.1	2	3.2	3.4	2.4	4.6	6	2.3	2.9	2.9	1.7	3.3	6
4	2.4	2.6	2.4	4.5	2.3	2.9	2.8	3.2	3.2	6.8	2.8	3.8	2.2	2.1	2.4	4.1	4	3.4	4.7	2.2	3.2	6.8
5	2.1	2.7	3.7	2.7	2.5	3.9	1.5	5.3	1	3.4	4.5	2.7	3.6	2.9	1.8	4	3.8	2.1	6.3	1.6	3.1	6.3
6	3.8	4.2	2.2	2.9	2.1	2.2	3	3.1	5.1	2.1	2.1	1.9	1.7	3	3.3	2.6	2	7.2	2.5	4	3.1	7.2
7	3	3.9	2.4	4	3.8	2.2	4.7	2.8	1.4	2.4	3.9	4.1	5.7	2	2.2	2.6	3.3	2.7	2.5	2	3.1	5.7
8	2.4	5.1	1.9	3.2	1.9	2.2	5.2	3.1	2.5	2.7	4.3	2.9	2.1	3.1	5.4	3.4	2.1	3.5	2.9	2.3	3.1	5.4
Avg	2.7	4.1	2.6	3.2	2.9	3.1	3.7	3.8	2.8	3.5	3.2	3.1	3.1	2.6	3.1	3.7	3.4	3.4	3.4	2.8	3.2	
Max	4	7.2	3.7	4.5	5.6	4.6	5.2	5.3	5.1	6.8	4.5	4.1	5.7	3.2	5.4	6	7.1	7.2	6.3	5.1		7.2

BDS Scenario

User ratings

STP1	STP2	STP3	STP4	STP5	STP6	STP7	STP8	STP9	STP10	LTP1	LTP2	LTP3	LTP4	LTP5	LTP6	LTP7	LTP8	LTP9	LTP10
Good	Acceptable	Acceptable	Very Good	Acceptable	Acceptable	Acceptable	Acceptable	Acceptable	Acceptable	Acceptable	Acceptable	Acceptable	Bad	Very Good	Acceptable	Good	Acceptable	Good	Bad

Opening Charts

Run	STP1	STP2	STP3	STP4	STP5	STP6	STP7	STP8	STP9	STP10	LTP1	LTP2	LTP3	LTP4	LTP5	LTP6	LTP7	LTP8	LTP9	LTP10	Avg	Max
1	1.7	1.8	1.5	2.1	2.6	1.8	2.2	2.4	1.5	2.1	2.8	2.3	3.8	2.7	2.5	1.2	1.5	1.6	1.7	2.3	2.1	3.8
2	2.8	2.9	1.9	3.8	1.3	1.1	1.3	2.1	1.8	1.1	1.8	2.3	2.3	1.8	1.6	1.2	1.9	1.5	1.6	2.9	2	3.8
3	1.7	2.8	2.6	2.8	2.1	2.8	3	2.5	2.3	2.1	3.5	2.5	1.6	2.7	2.1	2.2	1.2	1.3	2.3	2.1	2.3	3.5
4	1.5	1.5	2.9	1.8	1.8	1.5	1.7	2.7	2.7	2.2	2.2	1.8	1.1	2.9	2.1	2.5	2.1	2.4	2.9	2.7	2.2	2.9
5	2.8	2.9	2	1.3	2.4	1.4	2	2	1.2	3.2	1.3	4.1	3.4	3.1	2.8	1.8	2.5	1.7	1.5	3.2	2.3	4.1
6	2.1	3.2	2	2.1	2.6	2.4	2.5	2.2	2.9	2.5	1.6	2.3	2.6	2.7	2.9	1.5	3	1.8	2.2	2.4	2.4	3.2
7	2.7	2.2	1.8	2.8	2.6	1.9	1.4	3	2.7	1	2.3	1.4	1.3	1.9	2	2.9	1.3	2	2.8	1.8	2.1	3
8	1.5	3.3	2.1	1.8	3.6	1.2	1.2	1.2	1.9	2.4	2	2.1	2.2	2.4	2.6	2	3.8	2.2	2.5	3	2.3	3.8
Avg	2.1	2.6	2.1	2.3	2.4	1.8	1.9	2.3	2.1	2.1	2.2	2.4	2.3	2.5	2.3	1.9	2.2	1.8	2.2	2.6	2.2	
Max	2.8	3.3	2.9	3.8	3.6	2.8	3	3	2.9	3.2	3.5	4.1	3.8	3.1	2.9	2.9	3.8	2.4	2.9	3.2		4.1

UC 01

Run	User 1	User 2	User 3	User 4	User 5	User 6	User 7	User 8	User 9	User 10	User 11	User 12	User 13	User 14	User 15	User 16	User 17	User 18	User 19	User 20	Avg	Max
1	2	4.3	2.5	1.3	3	1.4	1.2	2.3	1.8	2.5	2.1	3.7	2.4	1	1.5	2	2.3	2.5	2.9	2.4	2.3	4.3
2	1.9	1.6	1.7	3.6	2.3	2.8	1.8	2.6	2.7	2.1	2.8	2.1	1.9	2.2	2.3	2	2.5	1.2	2.9	2.8	2.3	3.6
3	2.4	1.8	2.2	2.3	4.4	2.5	2.1	1.7	2	2.1	2.2	2.2	2.7	1.3	1.6	2.5	2.1	2.7	2.6	1.3	2.2	4.4
4	1.1	2.8	3.3	1.9	2	1	2.5	4.2	2.9	4.1	2.1	2.3	2.2	2.8	3.2	2	1.2	1.1	2.4	1.9	2.4	4.2
5	2.9	2.1	2.1	1.3	1.6	1	2.7	1.4	1.1	1.6	2.9	2.7	2.7	1.9	1.6	2	2.6	2.6	2.1	5	2.2	5
6	1.3	2.3	1.8	1.7	2.3	2	1.4	1.5	1.6	2.6	2.4	2	2.7	2.5	1.4	2	1.5	1.9	2.4	4.5	2.1	4.5
7	2.2	2	4.5	1.8	1.2	3.4	2.9	1.6	2.3	3.9	2.1	2.3	2	2	1.5	4.2	2.3	3.1	1.3	1.7	2.4	4.5
8	2.7	2	2.3	2.9	2.2	2.3	1.6	2.2	2.4	2.4	1.5	3.1	2.9	2.8	2.2	2	1.3	2.7	1.1	1	2.2	3.1
Avg	2.1	2.4	2.6	2.1	2.4	2.1	2	2.2	2.1	2.7	2.3	2.6	2.4	2.1	1.9	2.3	2	2.2	2.2	2.6	2.3	
Max	2.9	4.3	4.5	3.6	4.4	3.4	2.9	4.2	2.9	4.1	2.9	3.7	2.9	2.8	3.2	4.2	2.6	3.1	2.9	5		5

UC 04

Run	User 1	User 2	User 3	User 4	User 5	User 6	User 7	User 8	User 9	User 10	User 11	User 12	User 13	User 14	User 15	User 16	User 17	User 18	User 19	User 20	Avg	Max
1	2.7	3.8	4.5	5.2	2.8	3.1	4.5	4.9	4.3	3.4	4.5	4	7.2	2.7	6.5	3.4	4.1	4.3	3.4	4.5	4.2	7.2
2	9.9	2.9	2.3	3.3	5.9	2.3	4.2	8.3	3.7	4.9	3.7	5.7	5.8	4.4	2.1	3.9	3.9	3.8	4.5	5.3	4.5	9.9
3	2.7	3.2	6.5	2	9.3	5.6	4.8	2.6	6.6	3.8	7.3	3.1	3.1	4.5	4.6	5.3	4	2.6	8.1	3.9	4.7	9.3
4	4.3	7.2	7	7	2.8	4.2	3	3.7	3.9	3.5	3.2	4.2	3.5	8	8.4	3.5	5.2	4.9	8.1	2.7	4.9	8.4
5	3.2	3	4	4.4	6.7	3.2	9	4.6	3.5	4.5	2.8	8.8	3.1	4.9	4.2	3.8	3.5	9.8	3.2	6.2	4.8	9.8
6	2.1	4.3	2.4	8	3.8	3.3	3	6.4	3.4	3.3	9.2	4.7	7.2	2.1	6.6	8.2	6.1	2.9	6	6	5	9.2
7	3.9	3.2	4.1	3.9	6.8	3.2	2.4	4.9	3.6	3.8	6.9	3.1	3.1	7.9	3	9.3	4.6	5.1	4.3	4.2	4.6	9.3
8	2.3	5.8	3.7	4.7	3.1	6.7	2.6	4.3	4	3	3.5	3.2	4.3	3.4	6.2	4.8	5.1	2.1	8	4.9	4.3	8
Avg	3.9	4.2	4.3	4.8	5.2	4	4.2	5	4.1	3.8	5.1	4.6	4.7	4.7	5.2	5.3	4.6	4.4	5.7	4.7	4.6	
Max	9.9	7.2	7	8	9.3	6.7	9	8.3	6.6	4.9	9.2	8.8	7.2	8	8.4	9.3	6.1	9.8	8.1	6.2		9.9

GLS Scenario

User ratings

LTP1	LTP2	LTP3	LTP4	LTP5	LTP6	LTP7	LTP8	LTP9	LTP10
Good	Acceptable	Bad	Bad	Very Good	Good	Acceptable	Acceptable	Acceptable	Bad

Opening Charts

Run	LTP1	LTP2	LTP3	LTP4	LTP5	LTP6	LTP7	LTP8	LTP9	LTP10	Avg	Max
1	1.8	2.2	1.3	3.8	1.9	1.1	2.3	2	1.5	1.4	1.9	3.8
2	1.2	2.2	2.6	2.1	1.3	2.5	1.8	3.2	2.7	2.4	2.2	3.2
3	1.3	3.8	1.8	1.2	1	2.6	4.6	1.3	2.3	5.2	2.5	5.2
4	1.5	3.3	3.3	1.7	2.3	1.4	3	1.2	1.2	3.3	2.2	3.3
5	1.3	1.8	2.6	2.2	1.3	1.8	1.8	1.4	4.9	1	2	4.9
6	2	2.9	2.7	3	1.4	3.6	1.8	3.6	3.3	1.8	2.6	3.6
7	1.4	1.7	1.2	1.4	2	1	1.9	1.8	2.6	4.2	1.9	4.2
8	4.3	1.6	4.3	3.3	1.6	1.4	2.9	2.2	4.2	1.3	2.7	4.3
Avg	1.9	2.4	2.5	2.3	1.6	1.9	2.5	2.1	2.8	2.6	2.3	
Max	4.3	3.8	4.3	3.8	2.3	3.6	4.6	3.6	4.9	5.2		5.2

UC 01

Run	LTP1	LTP2	LTP3	LTP4	LTP5	LTP6	LTP7	LTP8	LTP9	LTP10	Avg	Max
1	1.8	1.9	3.5	1.3	4.2	1.6	1.6	2.9	2.6	1.6	2.3	4.2
2	3.4	1.5	4.3	2.2	1.1	2.7	1.2	3.9	2	1.6	2.4	4.3
3	1.7	3.3	2	1.1	1.2	2.4	3.4	2.6	1	4	2.3	4
4	1.7	1.3	3.6	2.9	1.3	1.7	2.4	1	2.8	3.6	2.2	3.6
5	1.9	2.1	1.7	1.3	2.3	2.6	1.2	4.6	1.4	2	2.1	4.6
6	1.5	1.6	1.8	3.4	3.1	2.7	4	1.5	2.4	1.5	2.4	4
7	2	2	1.7	4.3	1.8	4.1	2.5	3.3	1.6	1.6	2.5	4.3
8	1.6	1.3	4.2	3.9	4.2	1.1	3.2	4	3.8	2.6	3	4.2
Avg	2	1.9	2.9	2.6	2.4	2.4	2.4	3	2.2	2.3	2.4	
Max	3.4	3.3	4.3	4.3	4.2	4.1	4	4.6	3.8	4		4.6

UC 04

Run	LTP1	LTP2	LTP3	LTP4	LTP5	LTP6	LTP7	LTP8	LTP9	LTP10	Avg	Max
1	4	1.8	6.2	5.7	4	2.8	4.5	4.8	4.6	3.6	4.2	6.2
2	2.9	1.2	2.3	6.3	2.5	7.7	5.1	3.1	5.2	5.1	4.1	7.7
3	3.1	5.6	6	5.7	3.8	5.7	2.4	5.4	5.5	5.5	4.9	6
4	5.2	2.5	1.9	2.8	4.6	5.5	2.9	5.5	2.3	3.1	3.6	5.5
5	7.6	6.3	3.2	6.5	4	1.2	5.4	5.3	1.8	1	4.2	7.6
6	5.1	6.7	2.6	1.8	4.5	8.4	2.3	7.1	4.2	4	4.7	8.4
7	7.9	2.4	2.4	4.1	3.3	6.9	1.2	1.8	5.6	1.9	3.8	7.9
8	3.7	2.6	6.1	5.9	3	6	6.4	2.3	1.5	2.7	4	6.4
Avg	4.9	3.6	3.8	4.9	3.7	5.5	3.8	4.4	3.8	3.4	4.2	
Max	7.9	6.7	6.2	6.5	4.6	8.4	6.4	7.1	5.6	5.5		8.4