Vrije Universiteit Brussel – Belgium
Faculty of Sciences
In Collaboration with Ecole des Mines de Nantes – France
and
University of Twente – The Netherlands
2003

Prioritization of Requirements

A Thesis submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
(Thesis research conducted in the EMOOSE exchange)

By: Fabiana Mara Nogoseke

Promoter: Prof. Theo D'Hondt (Vrije Universiteit Brussel)
Co-Promoter: Prof. Mehmet Aksit (University of Twente)

Abstract

In this thesis, a process is described for prioritizing requirements in software development processes. Software development projects have to deal with many requirements at the same time. Moreover, projects are generally bounded by constraints like budget and time. To deal with this complexity, the project managers have to prioritize requirements effectively.
This thesis describes an application and assessment of a method called Analytic Hierarchy Process (AHP), which can be used to prioritize requirements in software development processes.
Although necessary, overall comparison of many requirements at the same time is a difficult task. One may compare two requirements, however, rather effectively. The main objective of the AHP method is to derive overall ranking from pair wise comparisons.
The prioritization may help in developing effective strategies for software development. In addition, stakeholders' input in the prioritization process increases the possibility that the final product is more accurate in implementing the desired properties.
To illustrate the approach, a tool has been built. This tool imports use case diagrams from a standard CASE tool. The tool offers facilities for pair wise comparison of use case diagrams. Based on these comparisons, the tool computes the overall ranking of the requirements.

Keywords: requirement, use case, priority, analytic hierarchy process (AHP).

Acknowledgements

First of all, I want to thank God for giving me so wonderful opportunity in my life.
Thanks to my family for giving me support to do this master during all this time, even in the difficult moments. Also, for the motivation that they always transmitted to me.
Thanks to Mehmet Aksit for supervising this work. Additionally to Joost Noppen for giving me some suggestions to the dissertation and Maurice Glandrup for giving me some advice about the project. Also remembers to all the people who helped me in Netherlands. Thanks to Jacques Noyé and all people, specially the professors, who in some way participated of EMOOSE.
Thanks to Carlos Alberto Maziero who gave me the opportunity to do EMOOSE master that has been a great and unforgettable experience.
Thanks to all the new friends I made during my staying in Europe.
Many thanks also to all my EMOOSE friends, we have spent so nice moments together. To Diego de Sogos, Francisca Munoz, Joao Del Valle, Jocelyn Simmonds, Kaiye Xu, Peter Ebraert and Yan Chen. To Angela Lozano and Carlos Noguera for having shared nice moments with me in Netherlands. To Cintia Fernandez, also considered as an EMOOSEr. And specially to Walter Werner for his friendship and companionship.
Finally, thanks to all my friends in Brazil that kept talking with me.
In spite of the difficult moments, hard work and the distance from home, I have had many happy moments here. All the EMOOSErs made this year a very happy year. I will always remember this period with very nice memories.

Fabiana Mara Nogoseke
August 2003

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 The Context

This thesis was assigned by the Twente Research and Education on Software Engineering (TRESE) [TRE02] group of the University of Twente. The group researches various aspects of software engineering such as analysis and design of architectures.

Software development projects normally need to deal with many requirements at the same time. Moreover, projects are generally bounded by constraints like budget and time. To deal with this complexity, the project managers have to prioritize requirements effectively.

The TRESE group is interested in evaluating software development methods through industrial projects. The problem of dealing with large requirements was identified within a project, which was carried out at the Philips premises [ZON03]. Within this project, an architecture design method SYNBAD (Synthesis-Based Software Architecture Design), which was also developed by the TRESE group, was applied. The experiences in applying the method were quite positive. However, dealing with a large set of requirements and prioritizing them was experienced as the major difficulty. It is our opinion that dealing with a large set of requirements is not a specific problem of SYNBAD. In principle, this problem, can be experienced in any large software development process. We therefore believe that the problem of having excessive number of requirements has to be addressed by defining effective prioritization techniques.

Our prioritization method is based on a decision making process called Analytic Hierarchy Process (AHP). AHP is a theoretically founded approach to rank a list of competing options based on pair wise comparison of the options.

Another issue related to prioritization is the consistency of the result. It is possible to introduce some sort of inconsistency in the ranking process. This is because options are compared pair wise but not as a total list. The AHP method also provides a means to give an inconsistency measure to the ranking process

1.2 The Problem Statement

In general, industrial software development projects have to cope with many requirements. However, it is difficult to deal with many requirements at the same time, and therefore, the software engineers have to - in some way - select and/or order requirements. Moreover, most projects are also bounded by constraints like budget and time. All these constraints require appropriate techniques to order requirements so that the complexity is reduced and most urgent requirements are fulfilled first.

Another tendency of development is progressive released versions. Currently, fulfilling budgets and deadlines is considered more important than carrying out the best architecture design and/or fulfilling the complete set of requirements. Effective prioritization of requirements can help in dealing with these problems. Hence, this is main objective of this Master Thesis project. Of course, the system has to satisfy a set of core requirements, otherwise the it may not work correctly. Important requirements cannot be and should not be skipped.

Requirements need to be analyzed according to their importance. Prioritizing requirements means comparing the competing requirements and assigning relative weighting values to them. Reducing the number of requirements simplifies the design and may result in avoiding unnecessary costs.

1.3 The Approach

This thesis describes an application and assessment of a method called Analytic Hierarchy Process (AHP), which can be used to prioritize requirements in software development processes.

Although necessary, overall comparison of many requirements at the same time is a difficult task. One may compare two requirements, however, rather effectively. The main objective of the AHP method is to derive overall ranking from pair wise comparisons.
Based on the priority values attributed for each requirement the software engineer may decide to consider part of the requirements, at least initially. However, the prioritization process should avoid missing important requirements.

The method is based on pair-wise comparisons; this means that the competing requirements are analyzed two by two. Weights are assigned to each comparison that represents the relative importance of a requirement with respect to the other requirement.
The criterion used in the comparison may be based on the importance of the requirement in realizing the system, considering two competing requirements each time.
The weights can be discussed within a group of stakeholders so that a more balanced result may be obtained. Another option is to compare different results and then to discuss possible conflicts.

This project assumes that requirements are represented by use cases. A hierarchy is used to represent the relationships or dependencies among the use cases.
Normally, in systems development projects, the activity of prioritizing requirements is carried out when the requirements are specified. But during the other phases of the project, new requirements may be emerged. This means that prioritization should be applied again potentially in every phase of the software development process. In this way, the information from the prioritization can always be used in the strategies for planning different software development phases.

The priority values are obtained from the judgements given, in general, by the customers. These judgments corresponding to the pair-wise comparisons. However, customers may make some mistakes in the judgements. Some inconsistency can be introduced in applying the method. Nevertheless, the consistency of the judgements can be determined by the proposed method.

A tool was developed to implement the approach. The tool brings facilities to the user such as inputting the necessary data and to making pair wise comparisons. The tool imports use case diagrams from a standard CASE tool, in our case, Rational Rose. The loaded use case diagram is parsed into the system in order to be utilized in the prioritization process. The result of the prioritization process, as well as the consistency indices, can be easily consulted by the user.

1.4 Outline of the thesis

This document is divided in 6 chapters. Chapter 2 first introduces use case driven and synthesis based methods for software development. This chapter also briefly summarizes the AHP method, which is used as a general prioritization technique in decision support.

Chapter 3 describes how the AHP method can be applied in prioritizing requirements in software development processes. A complete example illustrates the application of the approach.

The interpretation of the approach and the consistency of the obtained results are discussed in chapter 4. This chapter also presents how to deal with incomplete data.

Chapter 5 covers the tool architecture, design and functionalities.

Finally, chapter 6 contains the conclusions of this work and some future steps.

Chapter 2

Software Development Methods and the AHP Method as Prioritization Technique

2.1 Introduction

Software architecture is the overall structure of the software and the ways in which that structure provides conceptual integrity for a system [SHA95]. In other words, architecture is the hierarchical structure of program components (modules), the manner in which these components interact, and the structure of the data that are used by the components. In a proper architecture design, components represent the major system elements and their interactions [PRE97].

While designing architecture, the complexity and the size of specifications generally make the design a difficult process. . In addition, large or complex software specifications hinder fulfilling the project deadlines. Large and complex software means a large number of requirements present in the specification. This requires some action to deal with the specification. Requirements needed to implement the system and they are defined by the stakeholders. A stakeholder is a group or an individual that has a "stake" (something to gain or lose) as a result of the activities of a business. The specification of the system should contain all the necessary requirements [DAV93][WIE96].
Dealing with many requirements is a problem for every software development method. This chapter first discusses two kinds of software development methods and illustrates how prioritization can be added to these methods.

This chapter also describes the Analytic Hierarchy Process (AHP) method. The steps of the method are explained in section 2.4. The examples shown in this chapter illustrate the process and all its features.

2.2 Specifying Requirements with Use Cases

Requirements gathering is always the first step in any software analysis activity. This phase is generally called *Software Requirements Analysis*.
Requirements gathering can take the form of a meeting in which customer and developer meet to define basic system and software requirements.

Requirements analysis is a software engineering task that bridges the gap between system-level software allocation and software design. Requirements analysis enables the system engineer to specify software function and performance, indicate software's interface with other system elements, and establish constraints that software must meet. Requirements analysis allows the software engineer to refine the software allocation and build models of the data, functional, and behavioral domains that will be treated by software. Requirements analysis provides the software designer with models that can be translated in to data; architectural, interface, and procedural design. Based on these requirements, the software engineer (analyst) can create a set of scenarios that each identify a thread of usage for the system to be constructed. The scenarios, often called use cases, provide a description of how the system will be used. Finally, the requirements specification provides the developer and the customer with the means to assess quality once software is built [PRE97].

2.2.1 Software Requirements

Requirements can be defined as a specification of what should be implemented. They are descriptions of how the system should behave or of a system property or attribute. They may be a constraint on the development process of the system.

The IEEE Standard Glossary of Software Engineering Terminology [IEE90] defines a requirement as:

1. A condition or capability needed by a user to solve a problem or achieve an objective.
2. A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed document.
3. A documented representation or a condition or capability as in 1 or 2.

A complete understanding of software requirements is essential to the success of a software development effort. No matter how well designed or well coded, a poorly analyzed and specified program will disappoint the user and bring grief to the developer. Therefore, this phase of software development requires special attention in the well definition of the requirements. The prioritization is fundamental in this phase to select the requirements to be developed in the case when it is necessary, cases with many requirements. This initial phase is important to the well development of the project, dealing activities and objectives. The care in this moment brings benefits to the software quality.

2.2.2 Use Case methods

The most popular method of writing use case descriptions remains true to the ideas of Ivar Jacobson, inventor of use case modeling [JAC92]. Jacobson's method involves a set of entry and exit criteria called pre-conditions and post-conditions respectively, and a core criteria called the flow of events. The flow of events describes one set of interactions between the actors (users or external systems) and the system being

mapped out. The flow of events presents a single path through the system en route to a successful outcome [MIL01].

Use case describes a set of activities of a system from the point of view of its actors. The use cases represent the system specification, the client's whishes. The use case is used to define the behavior of a system and it describes generically the functionality of the product being built. This means the requirements of the project. Use cases are used as a contract between the customer and the developer. The use cases contain all the requirements defined by the user to the system. All the use cases together makes up the *user case model*, which describes the complete functionality of the system
It has all the complete set of functionalities that should be developed. Based on the use cases, the user can have the whole vision of the system. The software architecture is derived from the use case specification in use-case driven approaches.
Use-case driven means that the development process follows a flow, it proceeds through a series of workflows that derive from the use cases. Use cases are specified, designed and at the end they are the source from which the testers construct the test cases. A use case is simply a written narrative that describes the role of an actor as interaction with the system occurs.

The notation of the use case requires at least:
- **Unique name:** a name to identify the use case in the specification;
- **Description:** the textual description of the actions covered by the use case.

However, this information may be attached to a use case to make it clear:
- **Actors involved**: the actors that interact with the process;
- **Preconditions**: conditions that should be verified in order to the execution of the process;
- **Postconditions**: conditions verified after the execution of the process;
- **Invariants**: conditions that don't change during the execution of the process;
- **Non-functional requirements**: global constraints in the system;
- **Process description as activity diagram**: activity diagram is a dynamic diagram that shows the activity and the event that causes the object to be in the particular state [CHI03];
- **Exceptions, Error situations**: possible exceptions and error situations that can occur in the process;
- **Variations**: alternative courses in the use case.

An example of use case, with some of the possible information, is described in the sequence to demonstrate how the use cases are defined textually.

User Case Name: Cash Withdrawal
Description: The customer chooses cash withdrawal from the menu of possible transaction types. The customer chooses an amount from a menu of possible amounts. The system verifies that it has sufficient money on hand to satisfy the request. If not, it informs the customer and aborts the transaction. If so, it sends the customer's card number, PIN, chosen account and amount to the bank, which either approves or disapproves the transaction. If the transaction is approved, the machine dispenses the correct amount of cash and issues a receipt. The bank is notified whether or not an

approved transaction was completed in its entirety by the machine; if it is completed then the bank completes debiting the customer's account for the amount.
Precondition: The machine is ready.
Postcondition: The machine is ready.
Exceptions, Error situations: All disapprovals simply result in an error screen and the transaction is aborted. If the transaction is disapproved due to an incorrect PIN, the Incorrect PIN extension is executed.

The requirements are refined in use cases. A *use case* is defined as a sequence of actions that the system provides for *actors* [JAC99]. Actors represent external roles with which the system must interact. Actors and use cases together form the use *case model*. The use case model is meant as a model of the system's intended functions and its environment, and serves as a contract between the customer and the developers.
The *Use Case Model* is a diagram illustrating the scope of the application being built. The diagram contains actors (roles played by people or systems external to the application being built) and the services or functions they request from the application. The use case diagrams in this document don't show the actors because they are not considered in the process of prioritization, which is focused in the use cases.

Figure 1. Use Case representation.

This thesis utilizes Unified Modeling Language (UML) [OMG01] [RUM98]. The Unified Modeling Language (UML) is a graphical language for visualizing, specifying, constructing, and documenting the artifacts of a software-intensive system. The UML offers a standard way to write a system's blueprints, including conceptual things such as business processes and system functions as well as concrete things such as programming language statements, database schemas, and reusable software components. So, the use cases are represented as shown in Figure 1.

An example of use case diagram is shown in Figure 2. It contains use cases and an actor, and the arrows indicate the relationships between them.
The *Unified Process [JAC99]*, for example, applies a use-case driven architecture design approach. The Unified Software Development Process is *use-case driven*, *architecture-centric*, and *iterative* and *incremental*.
Use case driven – use cases are used for establishing the desired behavior of the system and for verifying and validating the system's architecture
Architecture centric – system's architecture is used for conceptualizing, constructing, managing, and evolving the system under development
Iterative and incremental – stream of executable releases; each release reduces the most significant risk to the success of the project
The process uses UML.

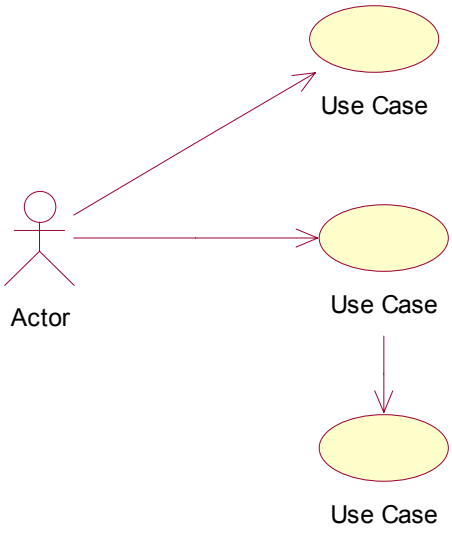


Figure 2. Use case diagram example.

2.3 The Synthesis Based Software Architecture Design Method (SYNBAD)

The idea of synthesis consists in that the initial problem is decomposed into sub-problems that are solved separately and later integrated in the overall solution [TEK00b]. In the synthesis process, the alternatives are searched and selected based on the existing solution domain knowledge.

The intent of *domain engineering* is to identify, construct, catalog, and disseminate a set of software artifacts that have applicability to existing and future software in a particular application domain. The overall goal is to establish mechanisms that enable software engineers to share these artifacts, to reuse them, during work on new and existing systems.

Software architecture design can be considered as a problem solving process in which the problem represents the requirement specification and the solution represents the software architecture design.
Software architecture methods based on synthesis are more complex then the methods only based on use cases.
In this method, requirements are transformed in problems. After that, the solution for the problems has to be found in the solution domain. Large number of requirements, problems and solutions are possible and pretty common in this kind of approach. The large number of these elements makes impossible or extremely difficult to cope with all of them in the development.

Synthesis based methods may require much more time per problem or requirement, because the solution is based on solution domain analysis, hence the case becomes more critical. On the other hand, this process may result in a more robust architecture, because the existing knowledge can be optimally utilized.

One main activity of the solution domain analysis process is the identification of the knowledge sources from which the necessary solution domain concepts will be extracted. If the set of possible solutions is very large, evaluating the knowledge sources may as such be complicated. In consequence, it may not be possible to examine all of them. For this reason we believe that prioritization is fundamental in this situation.

In section 3.2, we suggest an extension to the synthesis based approach so that problems can be prioritized and therefore the number of problems to be solved at a time can be reduced.

2.3.1 The phases of SYNBAD

The phases of SYNBAD are presented in Table 1 and are briefly described in the sequence.

Phase	Activities
1. Requirement Analysis	Specification of requirements Analysis of use cases and scenarios
2. Technical Problem Analysis	Generalization of requirements Identification of sub-problems Specification of sub-problems Prioritization of sub-problems
3. Solution Domain Analysis	Identification of solution domains Identification of knowledge sources Extraction of solution domain concepts Defines the conceptual structure
4. Alternative Design Space Analysis	Definition of alternatives for each concept Description of constraints
5. Architecture Specification	Extraction of architecture semantics Definition of dynamic behavior

Table 1. Phases and process of SYNBAD.

Figure 3 shows the phases of SYNBAD, which were presented in Table 1, and how the cycle works. This figure shows that from the Solution Analysis the cycle can go back to the Requirement Analysis. In Alternative Design Space Analysis, the alternatives are describes for each concept. These phases are more detailed in the following.

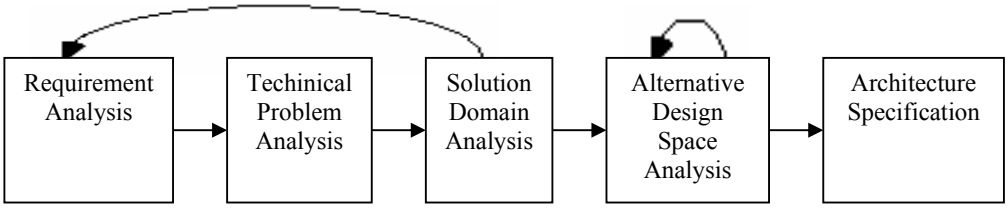


Figure 3. SYNBAD phases.

2.3.1.1 Requirement Analysis

Requirement Analysis is the first phase and it represents the specification that describes the requirements for the architecture to be developed. The basic goal is to understand the stakeholders' requirements. Stakeholders may be managers, software developers, maintainers, end-users, customers, etc. [HAE83] [TEK00a].
Informal specification of requirements is the basis to the requirement analysis. The synthesis-based design approach adopts the requirement analysis techniques such as textual requirement specifications, use-cases and scenarios, constructing prototypes and finite state machines. Figure 4 has the activities of this phase.

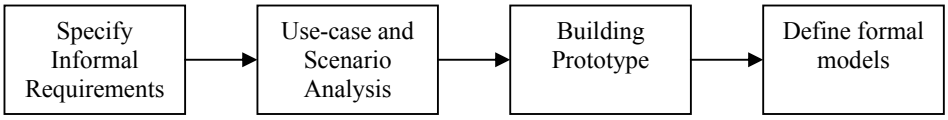


Figure 4. Synthesis-based Software Architecture Design Approach – Requirement Analysis.

2.3.1.2 Problem Analysis

In the Problem Analysis step, client requirements are abstracted and generalized. The idea is to identify the essence of the problem, separate from the client's view on the problem. The generalized requirements are mapped to technical problems. If necessary, the problems are decomposed into sub-problems. The technical problem is a general form of the requirement specification, and usually consists of several sub-problems. Identified technical problems are prioritized to their relevance before processing. This process has a basic way to prioritize the sub-problems, they are just ordered by the developer in the beginning, and it is not efficient. Now it can be improved with the method of our approach, described in section 2.4. Figure 5 has the activities of this phase.

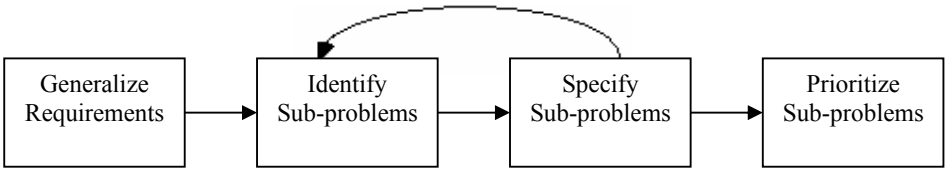


Figure 5. Synthesis-based Software Architecture Design Approach – Problem Analysis.

2.3.1.3 Solution Analysis

Basically, Solution Analysis represents the software architecture design.
For each sub-problem, the solution domains are searched to provide the solution abstractions to solve the technical problem.
If the solution domain knowledge doesn't exist, it can be abandoned or defined. In the first case, the problem is not solved due to lack of knowledge. The second case suggests a research to explore e formalize the concepts of the required solution domain.
Two factors are considered in the knowledge sources: objectivity and relevancy.
Objectivity defines the general acceptance to the knowledge source. The relevancy refers to the relevancy for solving the identified problem.
It is required that the solution domain knowledge is objective and relevant to be suitable for solving a problem.
These factors are utilized to define some priorities to the solutions domain. It is just an intuitive form to give some order to the solutions domain knowledge sources found to a certain problem. For solving the problem, fist the solution domain with the higher priority is utilized. The approach based on the AHP method presented in this work can improve this kind of prioritization. It can give more accurate and reliable results.

The solution domain knowledge may include a lot of knowledge that is covered by books, research papers, case studies, reference manuals, existing prototypes/systems etc. [TEK00a]. The solution domain concept is extract from kind of information.
The conceptual structure is defined with the solution domain concepts identified in the previous step are structured and refined.
Figure 6 has the activities of this phase.

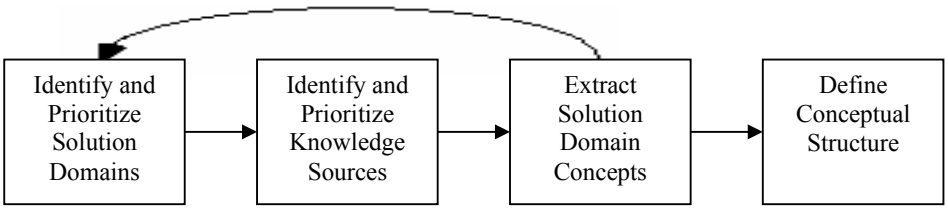


Figure 6. Synthesis-based Software Architecture Design Approach – Solution Analysis.

2.3.1.4 Alternative Design Space Analysis

The set of possible design solutions that can be derived from a given conceptual software architecture is called the alternative space. This phase has two sub-process *Defining the Alternatives for each Concept* and *Describing Constraints between Alternatives.*
In the *Defining the Alternatives for each Concept* sub-process, the alternatives are defined for each concept. The total set of alternatives of a concept may be very large, so it is necessary to identify the relevant alternatives and discard the other ones.
Architecture consists of a set of concepts that are combined in a structure. In the *Describing Constraints between Alternatives* sub-process, these concepts may be combined in many different ways. Many possible solutions can be found with these combinations. So, it is important to reduce the alternative space. The alternative space is reduced defining some constraints, and the relevancy from the client's perspective.
Figure 7 has the activities of this phase.

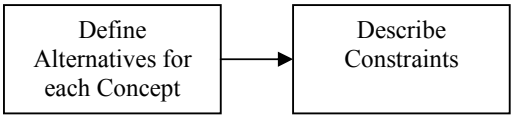


Figure 7. Synthesis-based Software Architecture Design Approach – Alternative Design Space Analysis.

2.3.1.5 Architecture Specification

The Architecture Specification process consists of two sub-process *Extracting Semantics of the Architecture* and *Defining Dynamic Behavior of the Architecture.*
The *Extracting Semantics of the Architecture* sub-process provides a more formal specification for each concept. The semantics of each concept are derived from the solution domain.
In the *Defining Dynamic Behavior of the Architecture* sub-process, the specifications of the architectural components are used to model the dynamic behavior of the architecture.
Figure 8 has the activities of this phase.

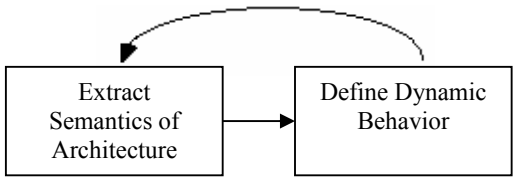


Figure 8. Synthesis-based Software Architecture Design Approach – Architecture Specification.

This is a general overview about SYNBAD method. More information is in [TEK00a] where SYNBAD is explained with more details and examples.

2.4 Analytic Hierarchy Process

The section introduces the Analytic Hierarchy Process (AHP) method, which has been developed to prioritize requirements in decision making.

2.4.1 Introduction

The Analytic Hierarchy Process (AHP) method was developed by Professor Thomas Saaty [SAA80] [GOL89]. This method is used for decision making and it has wide application in many areas like economic, social and management science [SAA93].
The theory was developed to solve a specific problem in contingency planning [SAA72] and a later major application was to design alternative futures for a developing country, the Sudan [SAA77a]. The result was a set of priorities and an investment plan for projects to be undertaken there in the late 1980's. The ideas have gradually evolved through use in a number of other applications ranging from energy allocation [SAA79], investment in technology under uncertainty, dealing with terrorism [SAA77b], buying a car to choosing a job.
The method is based on "pairwise" comparisons of alternatives. AHP represents a theoretically founded approach to computing weights representing the relative importance of criteria. The AHP has attracted the interest of many researchers mainly due the nice mathematical properties of the method.

2.4.2 Hierarchy

AHP structures the decision problem in levels to deal with complex decisions. In this way, the decisions are done in smaller sets of alternatives. The hierarchy is divided in levels.

In this approach, more abstracts requirements or use cases are put on the top of the hierarchy. At the bottom, the use cases are more specialized. We consider that abstracts requirements represent the most crucial requirements to the system because they contain the mainly activities or they are independent from other use cases. The other requirements are dependent on use cases.

Figure 9 illustrates an example of a hierarchy of use case diagrams. It is a use case diagram only to demonstrate the levels.
The lines show the relationships among the use cases. There are several kinds of relationships: *association*, *generalization*, *include* or *extend* relationships [OMG01].

Association: The participation of an actor in a use case; that is, instances of the actor and instances of the use case communicate with each other. This is the only relationship between actors and use cases.
Extend: An extend relationship from use case A to use case B indicates that an instance of use case B may be augmented (subject to specific conditions specified in

the extension) by the behavior specified by A. The behavior is inserted at the location defined by the extension point in B, which is referenced by the extend relationship.
Generalization: A generalization from use case C to use case D indicates that C is a specialization of D.
Include: An include relationship from use case E to use case F indicates that an instance of the use case E will also contain the behavior as specified by F. The behavior is included at the location which defined in E.

The relationship represents some sort of dependency among the use cases.
This example has a hierarchy with three levels. The requirements *Exchanging Data*, *Logging Information* and *Measurements* are in the first level. For instance, the use case *Reading Data* has an association with *Exchanging Data*.

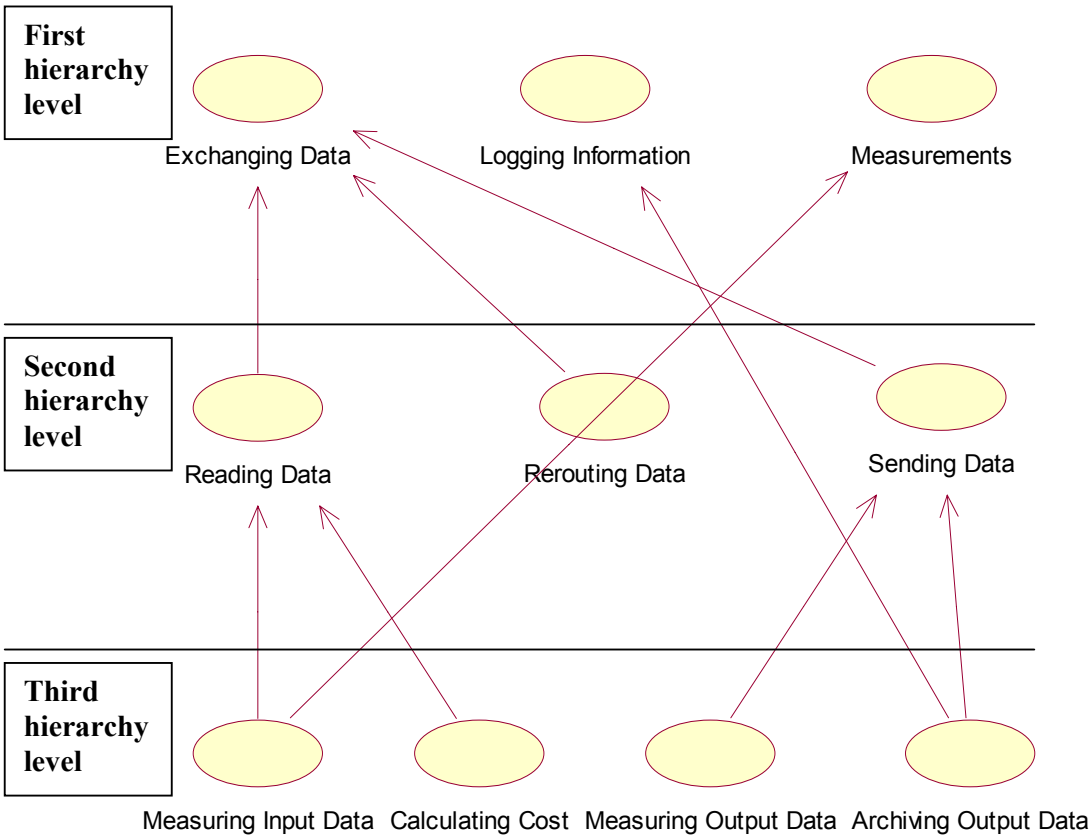


Figure 9. A hierarchy for priorities of use case diagram.

Priorities are calculated for each level. The priority values in the top level have influence above the other levels, as is explained in section 2.4.3.2.4. The first level, or top level, is the most important because it has the more general requirements, therefore the main decisions are done in this level to choose which branches in the hierarchy will be developed first.
One requirement can have one or more dependent requirements. However requirement doesn't have necessarily dependencies. Moreover, one requirement can be dependent on one or more requirements. Requirements can be dependent on requirements from different levels.

2.4.3 The Process

The AHP method provides a mathematical process to input subjective and personal preferences of an individual making a subjective decision. AHP has a sequence of steps in order to get the priority values. It consists in three main steps:

1. Form a relative priority matrix
2. Calculate the relative priorities
3. Determine the consistency of the results

All this steps are described in details in the sequence.

2.4.3.1 Pair Wise Comparisons

One of the crucial steps in decision making method is the accurate estimation of the data. This is crucial because there is the need to extract qualitative information from the decision-maker. It is very difficult to quantify data in terms of absolute values correctly. This method attempt to determine the relative importance, or weight, of the alternatives in terms of the importance criterion. Pair wise comparisons are used to determine the relative importance. In this approach, the decision-maker has to express his opinion about the value of one single pair wise comparisons at a time.
The first step of the method is to carry out the pair wise comparisons. This action requires a matrix, for displaying the comparisons, and the judgments, for implementing the comparisons.

	Req. 1	Req. 2	Req. 3	Req. 4	Req. 5	Req. 6
Req. 1						
Req. 2						
Req. 3						
Req. 4						
Req. 5						
Req. 6						

Table 2. Relative priority matrix of order 6.

First, a matrix of order *n*, where *n* is the number of elements or alternatives, is built. For instance, if there are six requirements, a matrix 6 x 6 is build. The matrix has to be filled with values defined in the judgmental process. The name or identification of the requirements is placed here to indicate the column and row to each requirement. The identification is placed in the first column and first row. Table 2 exemplifies a matrix 6 x 6.

2.4.3.1.1 Judgmental process

Pairwise comparisons are done with the alternatives, in our case the requirements or use cases. The pair wise comparisons should be based on the relative importance of the alternatives.
The judgments are used to fill the defined matrix. In addition, the judgments are done per level, this means that each level has its set of judgments. Only requirements in the same level are compared. The judgement process is described in the following subsection.

2.4.3.1.2 Estimating relative weights

It is in general very difficult to determine the most important requirements from a large set. Guessing the weight of each object directly, then comparing all the weights would be a hard task
For example, if there are 20 requirements, one has to decide first which one is the most important, then the second one and so on. Analyzing all the alternatives at the same time and deciding which are more important is not easy A practical alternative could be using pair wise comparisons.
The experiment requires more information, because it is necessary compare each object with all the other objects. But the result should be better, because there is a more precise analyze about the weights. For each comparison just the two objects in case are analyzed. In the case with no comparisons in pairs, it is necessary to think about all the objects in the same time to set the weights. This can be difficult with a great number of objects. For this reason the result using comparisons can be more precise.
The advantage is focusing exclusively on two objects at a time and thinking on how they relate each other. This process also generates more information to the method calculation.

Applying the comparison to the requirements, simply the element that is more important must to be chosen and how much more important it is. This choice is represented by the value of the comparison. The values utilized in the comparison are explained in the next paragraph.

2.4.3.1.3 Scale Comparison

The scale used for purpose is described in the Table 3. The scale consists in a range of values from 1 to 9.

Saaty describes in his book [SAA80] some experiments and conclusions about the numerical values used in the scale. Any scale could be used, like values between 1 and 20 or 1 and 100. However, it is much more difficult for users to give values using a large scale. In theory, any number less than infinity can be used for the upper bound. Extensive practical experience, however, suggests that 9 is a good upper bound to use.

Relative intensity	Definition	Explanation
1	Equal importance	Two elements are of equal value
3	Slightly more importance	Experience slightly favor one element over another
5	Essential or strong importance	Experience strongly favor on element over another
7	Very strong importance	An element is strongly favored and its dominance is demonstrated in practice
9	Extreme importance	The evidence favoring one over another is of the highest possible order of affirmation
9	Extreme importance	The evidence favoring one over another is of the highest possible order of affirmation
2, 4, 6, 8	Intermediate values between two adjacent judgements	When compromise is needed
Reciprocals of above nonzero	If the activity *i* has one of the above nonzero number assigned to it when compared with the activity *j*, then *j* has the reciprocal value when compared with *i*	A reasonable assumption
Rationals	Ratios arising from the scale	If consistency were to be forced by obtaining *n* numerical values to span the matrix

Table 3. Scale for pairwise comparisons.

Some reasons for setting 9 like the upper limit on the scale:

A scale from 1 to 9 has the advantage of simplicity.
The qualitative distinctions are meaningful in practice and have an element of precision when the items being compared are of the same order of magnitude or close together with regard to the property used to make the comparison.
It was noted the ability to make qualitative distinctions is well represented by five attributes: equal, weak, string, very strong and absolute. Compromises between adjacent attributes can be made when grater precision is needed. The totality requires nine values and they may well be consecutive.

Using a scale of pairwise comparison from 0 to ∞ may be not useful. As how is known from experiences, the ability to discriminate is highly limited in range and when there is considerable disparity between the objects or activities being compared, the guesses tend to be arbitrary and usually far from the actual. This suggests that the scale should have a finite range.

2.4.3.1.4 Judge matrix generation

The relative values are inserted in a matrix *n x n*, where *n* is the number of the elements. By convention, the comparison is always done with the element in the column, on the left, against an element in the row, on top.
For instance, in Table 2, the comparisons are done as the pairs: Req. 1 with Req. 2, Req.1 with Req. 3, Req. 1 with Req. 4, until the end of the first row. After that, the second row is evaluated and so on. This is a recommendation to the execution of the comparisons, to make it easier and ordered.

The number of comparisons is defined by:

$$n \cdot (n-1)/2 \quad (n \text{ is the number of elements}).$$

For instance, if there are 7 requirements to be compared: $7 \cdot (7-1)/2 = 21$
In an example with 7 requirements, 21 comparisons are necessary.

The matrix is filled with the values from the judgements that are explained in the next section.

2.4.3.1.5 Judgements

The judgement is to define which element is more important in each pair of requirements. The judgements are represented by values using the scale explained in paragraph 2.4.3.1.3.
For instance, comparing element A against element B, this is the judgement: “How strongly important is element A than element B?”. Table 4 has the possible answers, which can be applied for any case changing the elements “A” and “B” by the elements to be compared.

Judgement	Value
If A and B are equally important	Insert 1
If A is weakly more important than B	Insert 3
If A is strongly more important than B	Insert 5
If A is demonstrably or very strongly more important than B	Insert 7
If A is absolutely more important than B	Insert 9

Table 4. Applying the judgements.

The values between the presented options, 2, 4, 6 and 8 can also be chosen. They mean the intermediate value between the values in Table 4. For instance, if the answer to the judgement is considered between **weakly more important** and **strongly more important**, the value to be inserted should be **4**.
To answer the question comparing elements A and B, the value is inserted in the position (A, B) where the row of A meets the columns of B.
Considering that the answer to the question "How strongly important is element A than element B?" as "A is weakly more important than B", the value inserted in the position (A, B) is "3". Table 4 contains the result of this judgement in the correct position.

	A	B	C	D
A		**3**		
B				
C				
D				

Table 5. Value 3 inserted in the position (A, B).

An element is equally important when compared with itself, so where the row of A and column of A meet in position (A, A) we insert the value 1. The value 1 is insert for all the comparisons between an element with itself, thus the main diagonal of matrix consists of 1.

	A	B	C	D
A	**1**			
B		**1**		
C			**1**	
D				**1**

Table 6. The main diagonal is filled with 1's.

Only the upper part of the matrix must be filled with the relative comparisons. Table 7 shows where the judgements should be done and the values inserted.

	A	B	C	D
A	**1**			
B		**1**		
C			**1**	
D				**1**

Table 7. Upper part of the matrix.

Above the main diagonal is filled with the reciprocal values. The reciprocal is a divided by the correspondent value. In the position (B, A) where the row of B meets column of A, the value is the reciprocal of the position (A, B). For instance, if (A, B) = 3, the value of (B, A) = 1 / 3. Table 8 has examples of reciprocal values. The reciprocal should be done for the entire matrix.

	A	B	C	D
A	1	**3**	**1 / 2**	
B	**1 / 3**	1		
C	**2**		1	
D				1

Table 8. Example of reciprocal values.

2.4.3.2 Calculating the Priority Vector

The basic calculation in this method is define in two steps as follows:

- Normalize the matrix.
- Average the values in each row to obtain ratings.

The simple example in Table 9, with few requirements, shows how to apply the calculation.
In the example, there are three requirements: Requirement 1 (R1), Requirement 2 (R2) and Requirement 3 (R3). The values here simulate judgements just to illustrate the calculation.
Building the matrix 3x3 and estimating values, the following matrix is obtained:

	R1	R2	R3
R1	1	1/3	2
R2	3	1	5
R3	1/2	1/5	1

Table 9. Example – matrix 3x3

2.4.3.2.1 Normalize de matrix

In this phase, each column is normalized by adding its weighted values and by dividing each weight by this sum.

	R1	R2	R3
R1	1	1/3	2
R2	3	1	5
R3	½	1/5	1
Sum	**4.5**	**1.53**	**8**

Table 10. Sum of the columns.

Table 10 shows the sum of the columns. After that each weight is divided by its respective sum of its column. The result of this normalization is shown in the next matrix:

	R1	R2	R3
R1	0.22	0.22	0.25
R2	0.67	0.65	0.62
R3	0.11	0.13	0.12

Table 11. Normalized columns.

2.4.3.2.2 Average the values in each row to obtain ratings

In the next step, the values in the rows are added.

	R1	R2	R3	**Sum**
R1	0.22	0.22	0.25	**0.69**
R2	0.67	0.65	0.62	**1.94**
R3	0.11	0.13	0.12	**0.37**

Table 12. Row sum.

Then making the average, dividing each row sum by the number of requirements.

$$1/3 \cdot \begin{pmatrix} 0.69 \\ 1.94 \\ 0.37 \end{pmatrix} = \begin{pmatrix} 0.23 \\ 0.65 \\ 0.12 \end{pmatrix}$$

This result vector is the vector of priorities. The result is an estimation of the eigenvalues of the matrix.

2.4.3.2.3 Vector of Priorities

The values obtained from the comparison matrix, after the execution of the steps, can be viewed as the following information:

Requirement 1	23 %
Requirement 2	65 %
Requirement 3	12 %
Total	100 %

Table 13. Priority values.

These values represent the percent value of the requirement's total value. The sum of all the result values is 100 percent. This means the priority of each requirement, which requirement is more relevant to the system. The result tells not only which requirement is more important, but it also brings the information about how much the requirement is more important in the set. This information is useful in the analyze, it helps to identify the essential requirements. Therefore, the values are more relevant in the study of the requirements with the lower priorities. These values can be analyzed and it is possible to verify how far from the other requirements the less important requirements are located. Those values are valuable in the decisions.

2.4.3.2.4 The dependencies

The judgments are done only at the first level, the priority to the other levels is calculated based on the dependencies. For instance, if a use case depends on a certain requirement, the value that is used to fill the matrix in this case is the priority value of this requirement. If there are more than one use cases dependent on a requirement, the value is proportionality shared among the use cases.
Looking at this example, this process is easier to understand:

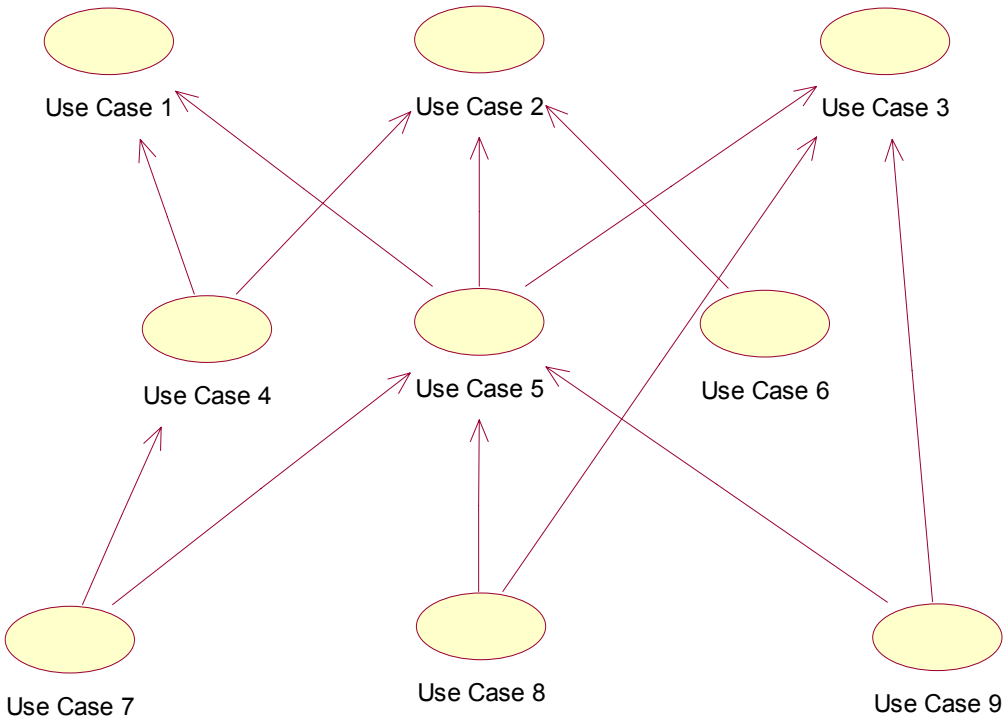


Figure 10. Example of dependence.

Figure 10 has an example with nine use cases organized in a hierarchy. This hierarchy was built only for illustrate how to define the values to the dependencies.

	Priorities
Use Case 1	23
Use Case 2	65
Use Case 3	12

Table 14. Priority values example to the first level.

This example assumes that the priorities to the first level have already been calculated the result is shown in Table 14. Defining the values to the level two the following process should be followed:

Use Case 1 has 2 associations, so its priority is divided by 2.

$$V_{UC1} = \frac{P_{UC1}}{2} = \frac{23}{2} = 11.5$$

Use Case 2 has 3 associations, so its priority is divided by 3.

$$V_{UC2} = \frac{P_{UC2}}{3} = \frac{65}{3} = 21.67$$

Use Case 3 has also 3 associations, so its priority is divided by 3.

$$V_{UC3} = \frac{P_{UC3}}{3} = \frac{12}{3} = 4$$

Use Case 4 depends on *UC1* and *UC2*. So, the values are added.

$$V_{UC4} = V_{UC1} + V_{UC2} = 11.5 + 21.67 = 33.17$$

Use Case 5 depends on *UC1*, *UC2* and *UC3*. So, the values are added.

$$V_{UC5} = V_{UC1} + V_{UC2} + V_{UC3} = 11.5 + 21.67 + 4 = 37.17$$

Use Case 6 depends on *UC2*. So, the value is attributed.

$$V_{UC6} = V_{UC2} = 4$$

With the values defined, the matrix is built:

	Use Case 4	Use Case 5	Use Case 6
Use Case 4	1	33.17 / 37.17	33.17 / 4
Use Case 5		1	37.17 / 4
Use Case 6			1

Table 15. Matrix for level two.

Applying the method, the calculation is the same from the first level. The reciprocal values are also identified, before starting the normalization.

	Priorities
Use Case 4	45
Use Case 5	50
Use Case 6	5

Table 16. Priorities to the second level.

Table 16 shows the priorities found to the second level. To calculate the priorities to the third level, the process executed to the second level must be executed in the same way. The values derived from the priorities in the previous levels are defined and the matrix is built based on the dependencies. This values helps to visualize the use cases dependent on the most important requirements in the first level. This can be used to define de order of developing. Use cases in these levels will be skipped if they are dependent on some use case removed from the specification.
As the judgements are done to the fist level, the hierarchy in this case helps to reduce the number of necessary relative comparisons.

2.4.3.3 Calculating the Consistency Value

The consistency index is to indicate if the result is consistent or not.

Being consistent means that with a basic amount of raw data, the other data can be logically deduced from it. In pairwise comparison of *n* elements, with *n* –1 pairwise comparison judgments the other judgments can be deduced. The relation used to deduced is doing the following:

If Requirement 1 is 3 times more important than Requirement 2 and Requirement 2 is 6 times more important than Requirement 3 then $R_1 = 3R_2$ and $R_1 = 6R_3$. It should follow that $3R_2 = 6R_3$ or $R_2 = 2\ R_3$ and $R_3 = ½R_2$.
If the value of the judgment of R2 and R3, position (2, 3) in the matrix, is different from 2 then the matrix would be inconsistent.

The consistency is equivalent to the number of requirements that is as the maximum eigenvalue λmax.

2.4.3.3.1 Consistency index

The consistency index is an indicator of result accuracy of the pairwise comparisons.
λmax denotes the maximum principal eigenvalue of the comparison matrix.
The departure from the consistency is the difference *λmax – n* divided by *n – 1*, defined by:

$$CI = \frac{(\lambda \max - n)}{(n-1)}$$

Using the values from the previous example, the consistency index is obtained as shown in the following steps:
Estimating *λmax,* first the comparison matrix is multiplied the by the result vector:

$$\begin{pmatrix} 1 & 1/3 & 2 \\ 3 & 1 & 5 \\ 1/2 & 1/5 & 1 \end{pmatrix} \cdot \begin{pmatrix} 0.23 \\ 0.65 \\ 0.12 \end{pmatrix} = \begin{pmatrix} 0.69 \\ 1.95 \\ 0.37 \end{pmatrix}$$

Then the first element of the resulting vector is divided by the first element in the result vector, the second element of the resulting vector by the second element in the result vector, and so on:

$$\begin{pmatrix} 0.69 & / & 0.23 \\ 1.95 & / & 0.65 \\ 0.37 & / & 0.12 \end{pmatrix} = \begin{pmatrix} 3.00 \\ 3.00 \\ 3.08 \end{pmatrix}$$

Calculating λmax, average over the elements in the resulting vector:

$$\lambda \max = \frac{3.00 + 3.00 + 3.08}{3} = 3.03$$

Finally, the consistency index:

$$CI = \frac{\lambda \max - n}{n - 1} = \frac{3.03 - 3}{3 - 1} = 0.015$$

The closer the value of λmax is to n, the smaller the judgmental errors.

2.4.3.3.2 Random Indices

The consistency can be compared with its value from random indices.
The values in the random indices are obtained from randomly chosen judgments and corresponding reciprocals in the reverse positions in a matrix of the same size.
Comparing the value of the consistency index with its random indices, how bad the consistency may be in a given problem can be estimated.
The consistency indices of these random judgments are represented in Table 17. The first row represents the order of the matrix and the second one is respectively the consistency index.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0,00	0,00	0,58	0,90	1,12	1,24	1,32	1,41	1,45	1,49	1,51	1,48	1,56	1,57	1,59

Table 17. Random indices.

This table was generated applying the method to a large number of matrices of the same order. The matrices received randomly generated judgments and the corresponding reciprocals. The consistency index was defined for all the matrices and then the average of these consistency indices was calculated.

2.4.3.3.3 Consistency ratio

The ratio of the Consistency Index to the average the Random Indices for the same order matrix is called the Consistency Ratio (CR). The value of CR is defined dividing the CI by CR:

$$CR = \frac{CI}{RI}$$

Calculating CR for the example, the value of RI is found in Table 17. According to this table, the RI is 0.58 for matrices of order 3, as in the example.
Thus, the consistency ratio for this example is:

$$CR = \frac{CI}{RI} = \frac{0.015}{0.58} = 0.026$$

The rule applied to the Consistency Ratio says that if CR is more than 0.10, this means that there are inconsistencies in the ratings.
In the example the value of CR is 0.026, that means that the result is consistent, therefore the comparison matrix has consistent judgments.
However, consistency ratios exceeding the value of 0.10 are common.
The other levels have 0.00 to the consistency ratio, because the values used to fill the matrix are calculated as explained in section 2.4.2.2.4. So, there is no inconsistency in these judgement because they are deduced from the previous level.

2.4.3.3.4 Revising Judgements

Based on the index the values given to the comparison may be reviewed and new priorities can be calculated.

Changes in priority at lower levels affect the priority of elements in lower levels. The lowest level of the hierarchy must prioritize very carefully because since these priorities drive the rest of the hierarchy.
The next example illustrates a not so good consistency ratio:

	Requirement 1	Requirement 2	Requirement 3	Requirement 4
Requirement 1	1	1/3	2	4
Requirement 2	3	1	5	3
Requirement 3	1/2	1/5	1	1/3
Requirement 4	1/4	1/3	3	1

Table 18. Judgment Matrix.

Applying the method the following vector with the priorities is obtained:

	Priorities
Requirement 1	0.26
Requirement 2	0.50
Requirement 3	0.09
Requirement 4	0.16

Table 19. Priority Vector.

Calculating the consistency, the value of λmax is 4.37. The consistency index is 0.12. Finally, the consistency ratio, dividing the CI by 0.90 that corresponds to matrices of order 4, according to Table 17:

$$CR = \frac{CI}{RI} = \frac{0.12}{0.90} = 0.14$$

The consistency ratio of 0.14 indicates that the result is not so good like the expected. To improve this result, the judgments should be reviewed.

There is the possibility to find the inconsistency judgment by deduction. For instance, supposing that the first diagonal above the main diagonal is correct, the other values can be deduced. The deduced values can be compared with the values in the matrix.

The position (1, 3) can be deduced from (1, 2) and (2, 3):
If $R_1 = 1/3R_2$ and $R_2 = 5R_3$, then $R_1 = 5/3R_3$.
The matrix has the value 2 in this position. The difference is 2 – 5/3 = 0.33.

The position (2, 4) can be deduced from (2, 3) and (3, 4):
If $R_2 = 5R_3$ and $R_3 = 1/3R_4$, then $R2 = 5/3R_4$.
The matrix has the value 3 in this position. The difference is 3 – 5/3 = 1.33.

The position (1, 4) can be deduced from (1, 3) and (3, 4):
If $R_1 = 2R_3$ and $R_3 = 1/3R_4$, then $R1 = 2/3R_4$.
The matrix has the value 4 in this position. The difference is 4 – 3/2 = 3.33. In this case the difference value is bigger than difference values in the other two positions.
Trying to reduce the inconsistency the next matrix has the value in the position (1, 4) modified.

	Requirement 1	Requirement 2	Requirement 3	Requirement 4
Requirement 1	1	1/3	2	3/2
Requirement 2	3	1	5	3
Requirement 3	1/2	1/5	1	1/3
Requirement 4	1/4	1/3	3	1

Table 20. Modified judgment matrix.

Applying again the method, the new results are describe below:

	Priorities
Requirement 1	0.16
Requirement 2	0.52
Requirement 3	0.09
Requirement 4	0.23

Table 21. Priority vector.

Calculating the consistency to the modified matrix, the value of λmax is 4.03. The consistency index is 0.01.
Finally, the consistency ratio:

$$CR = \frac{CI}{RI} = \frac{0.01}{0.90} = 0.01$$

The consistency ratio improved considerably. The value has reduced from 0.14 to 0.01. This means that the errors in the judgments were reduced. The new consistency index of 0.01 means that the result is consistent.
It is not so easy to find the inconsistencies in the judgment matrix, but the deductions can help to guess the main errors. Even if after some modifications in the judgment values the consistency ratio is not acceptable, new modifications may be done. There is no limit, the values can always be modified in order to get the best result.
The method must be applied again after modifications. Then the result is analyzed as like the consistency ratio.

Chapter 3

Applying AHP to Prioritizing Requirements in Software Design

Introduction

This chapter describes how prioritization process can be introduced in the SYNBAD method. We discuss where the prioritization can be applied in this kind of software development. An example of software architecture specification, defined as *Message Exchange System*, is presented to demonstrate the method, emphasizing the prioritization process and the results. Section 3.4 introduces this example.

3.2 Adding Prioritization Techniques to the SYNBAD Method

The synthesis process can be summarized as in Figure 11. This figure shows the main activities in the process involving synthesis, these processes were explained in chapter 2, where the phases are described.

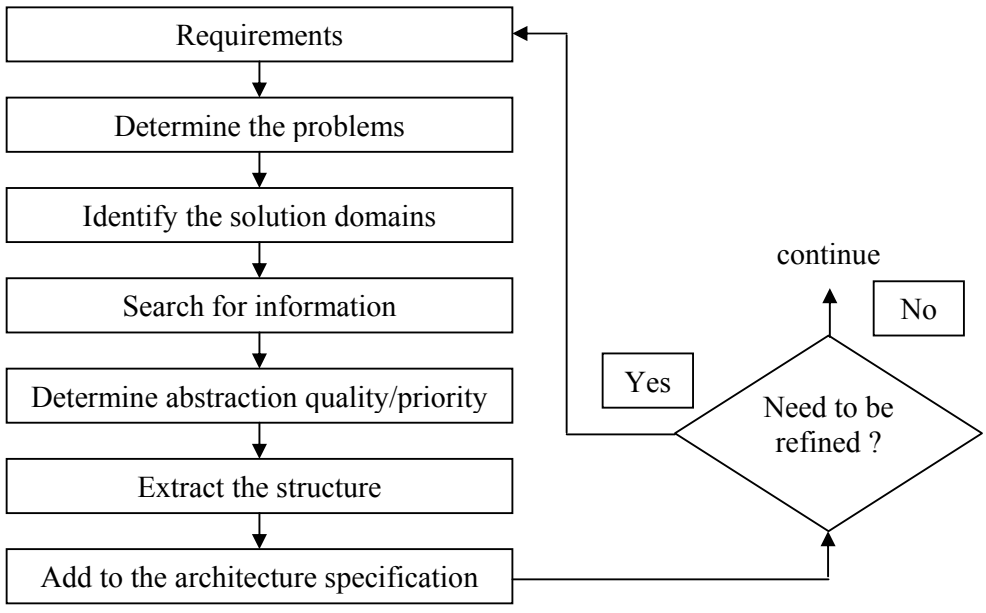


Figure 11. A synthesis process.

After definition of requirements, the problems are determined. Based on the problems the solution domains are identified. The information is searched in the solution domain. The abstraction of the solution is determined considering the features of quality and priority. Based on the information abstraction, the structure is extract. Finally, the structure is added in the architecture specification. There is still the possibility of refined the architecture, when new requirements can arise and the cycle starts again.

The synthesis process has different approaches to execute the activities. Here we describe the top-down and bottom-up scenarios.
Top-down scenario represents the dominating solution domain. The process goes from dominating domain to sub-domains. The dominating solution domain has a common solution for the problems. A structure is extract from the domain and abstractions are defined in this structure. Each abstraction has a set of sub-problems to solve. Figure 12 illustrates this scenario.

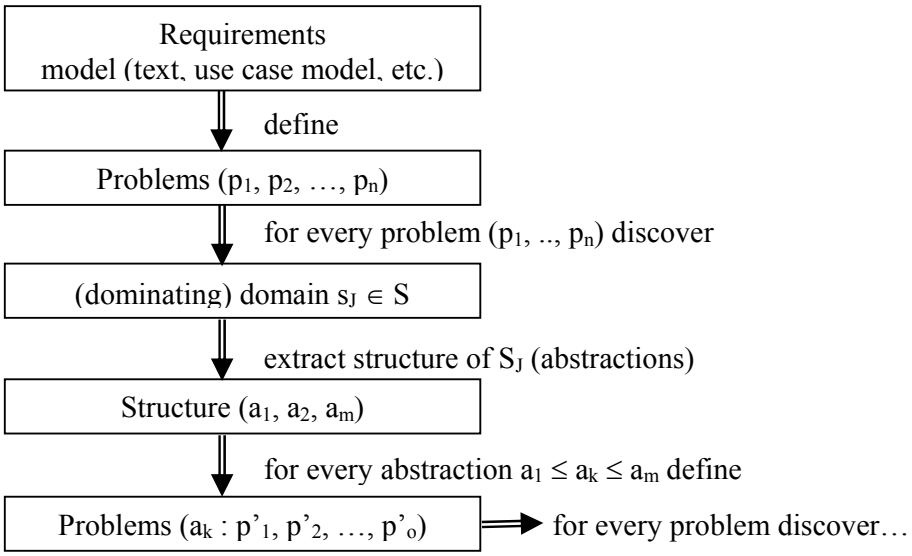


Figure 12. Top-down scenario.

Bottom-up scenario means discovering sub-solutions domains. The process goes from sub-domains to architecture. For each problem, a solution is discovered in the solution domain. A structure is extract from the solution domain for every solution. The structure contains the abstractions to the solutions. Each abstraction has a set of sub-problems to solve. Figure 13 illustrates this scenario.

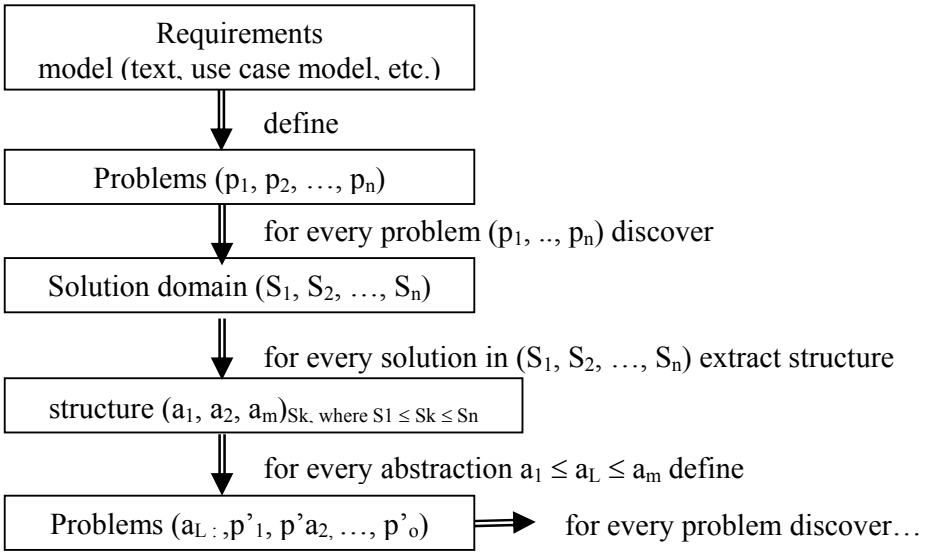


Figure 13. Bottom-up scenario.

3.2.1 Depth-first and Breadth-first

Strictly speaking, the flow of the process in SYNDBAD can be depth-first or breadth-first. These flows are different in the order of the activities, specially in the treatment of problems. Each problem consists usually of several sub-problems. The technical problem is a more general form of the requirement specification. So, the technical problem is decomposed into sub-problems, which are more specific then the problem, in order to solve it.

In the depth-first approach, first the problem is refined in all levels, until the last one. After that the other problems are defined. Technical problem analysis and solution domains are executed for each problem separated. Much time is necessary to this kind of application. A problem has to be completely defined, with all the sub-problems identified and specified, to start defining the next one. This is done until the last problem.

The other way, breadth-first approach says that first all problems are defined on a level. After that, the other levels are refined. Problems are refined level by level together. The same process happens with the solutions, everything is done level by level.

Figure 14 illustrates these aproaches.

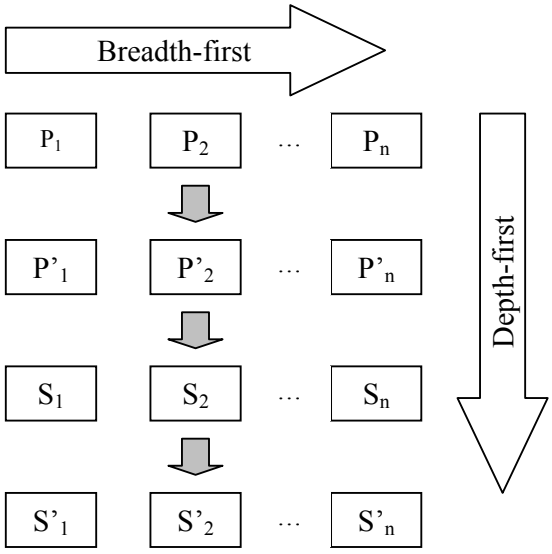


Figure 14. Breadth-first and Depth-first approaches.

3.3 The SYNBAD Process with Prioritization

SYNBAD has 5 phases as explained in chapter 2, however the two last phases, Alternative design space analysis and Architecture specification, are not mentioned here; for simplicity these phases are considered out of scope. The prioritization will be considered for the other three phases: Requirement Analysis, Technical Problem Analysis and Solution Domain Analysis.

Figure 15 shows the phases of the software architecture development of the method SYNBAD where the prioritization is applied. The elements in this figure are:
Requirements refers to the Requirement Analysis phase.
Problems refers to the Problem Analysis phase.
Solutions refers to the Solution Analysis phase.
Prioritizing indicates the process of prioritization.
Architecture means the software architecture obtained after all the phases.

The arrows show the possible directions and alternatives in the development. The cycle starts in Requirements, after defining the requirements the prioritization is applied and the cycle goes on with the Problems. After Problems Analysis there are two possibilities: new requirements can bee defined or the cycle continues through the solutions. Before starting to solve the problems, they are prioritized. In Solution Analysis there are three possibilities: new requirements can be defined, new problems can be defined or the solution is complete. The solutions are also prioritized to define the order of execution. Finally the architecture is defined, this involves the other two phases: Alternative design space analysis and Architecture specification.

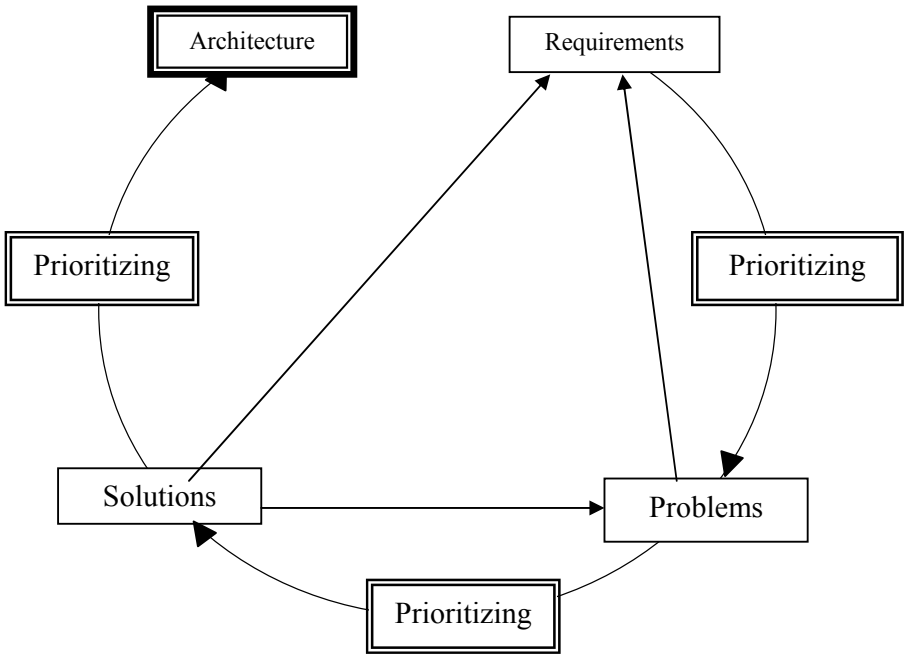


Figure 15. Phases and processes where priorities are defined with all the possible alternatives.

The initial prioritization of requirements is crucial to the software development, because in this moment the first actions in the project can be done accordingly with the strategies. The strategies to the project are planned after requirements were defined in order to specify the development schedule. The development schedule will define the activities in the project. So, the priorities of the specification will help to guarantee to reach the goal of the project. The initial prioritization will guide the start of the project.
The initial prioritization of problems is also crucial as the requirements. One requirement can be transformed in one or more problems and a problem can correspond to one or more requirements. Therefore, it is necessary to prioritize the problems to this new phase of the project. With the prioritization of the problems, the strategies to this phase can be prepared.

3.3.1 Depth-first and Breadth-first with Prioritization

This section describes how to apply the prioritization method to the depth-first and breadth-first strategies.

In the depth-first approach more time than the expected can be spent to solve each problem per time. In this case the problems can be prioritized in the initial level, then the order of development can be defined. But only the problems in the first level can be selected, because the other levels are defined case by case. The solutions can not be chosen, because they also are define per each problem and not at the same time.

In the breadth-first approach, when all problems are defined in the abstraction level, they can be prioritized, and problems are chosen to be refined in the next level depending on the priorities. The same process is possible to apply to solution domains. The solution domains in the abstract level are prioritized before starting to refine the next level. In this way, the number of problems and solutions to refine is reduced. If there is a great number of problems and solutions, with the prioritization and less elements to refine, the time to development is more guaranteed. The priorities assure that the most important problems and solutions to the project are developed.

In both cases, the selection and order for solving sub-problems may have impact on the final solution. Therefore, the weight of the priorities requires attention.

3.4 Application of SYNBAD to an Example Problem

As an example, a software architecture called *Message Exchange System* is considered. This example will be used to illustrate our approach.

Companies have the necessity of exchanging data. The communication medium is used in the transmission of the data. Concepts used in this example are described in Appendix B.
First the data in the communication medium has to be charged for their usage. The charged data is manipulated in same way to attend the clients. After that, the data is sent to the receiver. Figure 16 illustrates this flow of data.
Therefore, an exchange system has to be designed to attend this demand of the companies.

Figure 16. Exchange System.

The exchange system needs three basic operations: charging, interpreting and sending data. Figure 17 represents these needs.
So, the following architecture to an exchange system is considered:

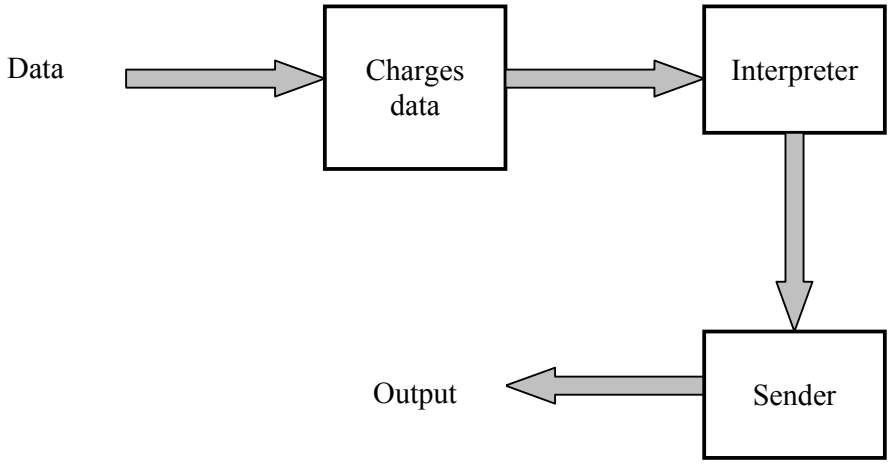


Figure 17. Data exchange process.

The *Message Exchange System* example is used to demonstrate the problem of selecting the subset of requirements for development. The system can not be developed in time with all this requirements, so it is necessary to choose a subset of requirements to attend the main client's wishes. So, the goal of this example is to fulfil the basic necessities of the system and to reduce the activities in the development.
In order to execute this task, the alternatives, which are requirement candidates, will be analyzed two by two. This system has the aim of exchange messages, so the focus is on the requirements responsible to carry out that. They are crucial to the result of the system development. More details about the criteria used in the analysis of the requirements are in section 3.8 where the pair wise comparisons are discussed for this example.
This necessity of reducing the number of requirements represents the problems found generally in software projects.

3.5 General Description of the System

This system has a certain number of requirements that should be developed. In the sequence, the basis of the specification is discussed.

The system has eleven basic requirements that represent the main functionalities in the system. They are briefly described as follows:

1. Reading data: reading messages to the system. This means to receive the messages from the clients. All the activities responsible to receive a message and charge the data are included in this requirement.

2. Converting data: converting the messages in the system. This means to transform the charged message in the standard format. All the activities responsible to converting the message are in this requirement.

3. Rerouting data: rerouting the messages in the system. This means to reroute the message to the right client. All the activities responsible to reroute the message are in this requirement.

4. Sending data: sending the messages. This means to send the message to the client. All the activities responsible to send the message are in this requirement.

5. Logging information: logs information during execution. Relevant information to the system is registered for future analysis or measurements. Everything that is logged is related to this requirement like input data, output data and errors. These are the relevant information considered.

6. Measurements: measures data and information in the system. This requirement includes the activities to measure the data and information in different stages of the process. The measurement can be applied to the logged data or data during the execution. The data can be the input, converted and output data, and also the information generated by the system during the execution, like error messages. Examples of measurements are number of messages, size of messages and number of errors. It can be done per client or per determined period of time.

7. Errors Control: controls the errors that happen during execution. This requirement covers the error handling defined in the system. It can define some action to a determined kind of error, for instance. Error handling can be executed in all the stages: in the beginning, with the input data; in the middle of the process, interpreting or converting data and in the end, sending data.

8. Statistics: applies statistics to the data in the system. The statistics can be applied to the logged or measured data. This requirement represents all kind the statistics that are utilized in the system based on the defined parameter. This is for instance: per determined period of time, per client, per kind of error.

9. Monitoring: monitors the information in the system. Logged information, error handling and measurements can be monitored. These activities can be visualized and analyzed as describes this requirement.

10. Defining Formats: refers to the message formats used in the exchange. The system can deal with different kinds of formats. This requirement defines the utilized in the system. And also how to insert new formats.

11. Defining Protocols: refers to the protocols used in the exchange. The system can deal with different kinds of protocols. This requirement defines the protocols utilized in the system. And also how to insert new protocols.

3.6 Requirement Specifications

Requirements of the *Message Exchange System* are represented by use cases. The specification of the system has a several number of use cases that are described in Appendix A. *A*ll the requirements and use cases defined to the *Message Exchange System* are represented in a use case diagram and use case models are explained in this section. Figure 18 shows the use case diagram with the main requirements, which were described in the previous section.

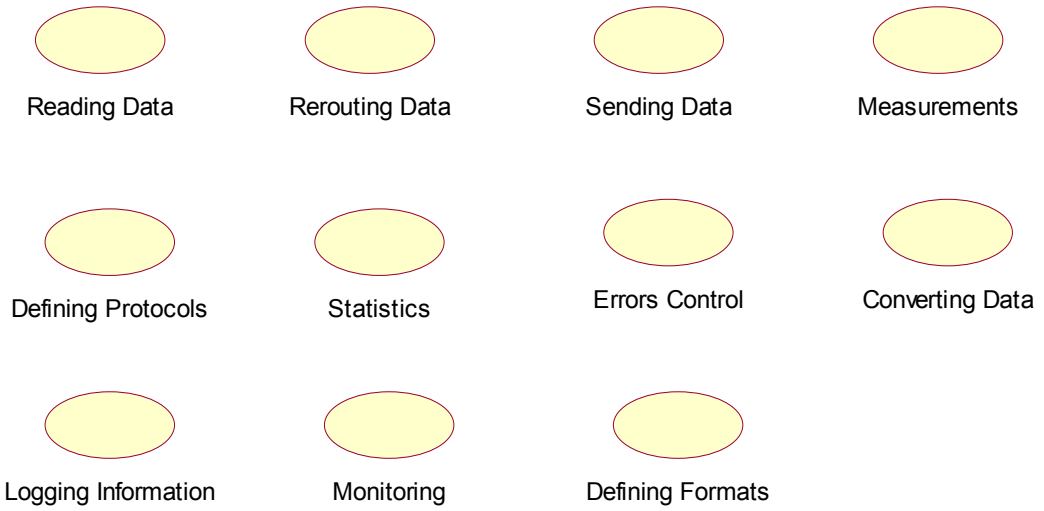


Figure 18. *Message Exchange System*: main requirements use case diagram.

Use cases are divided in five modules based on its main activity in order to facilitate the organization.
Figures 19 through 23 show the modules with all the use cases. Each figure represents a module. The requirements are inserted in the module depending on their definition.
The modules are:

Module 1: Processing Data
This module has the use cases responsible by exchanging the messages. It includes receiving, processing and sending the message. This module has the main use cases Reading Data, Sending Data, Rerouting Data e Converting Data. And it has also the auxiliaries use cases Buffering Data, Finding Format, Filtering Data, Calculating Cost, wrapping Data, Formatting Data and Output Data Validation.

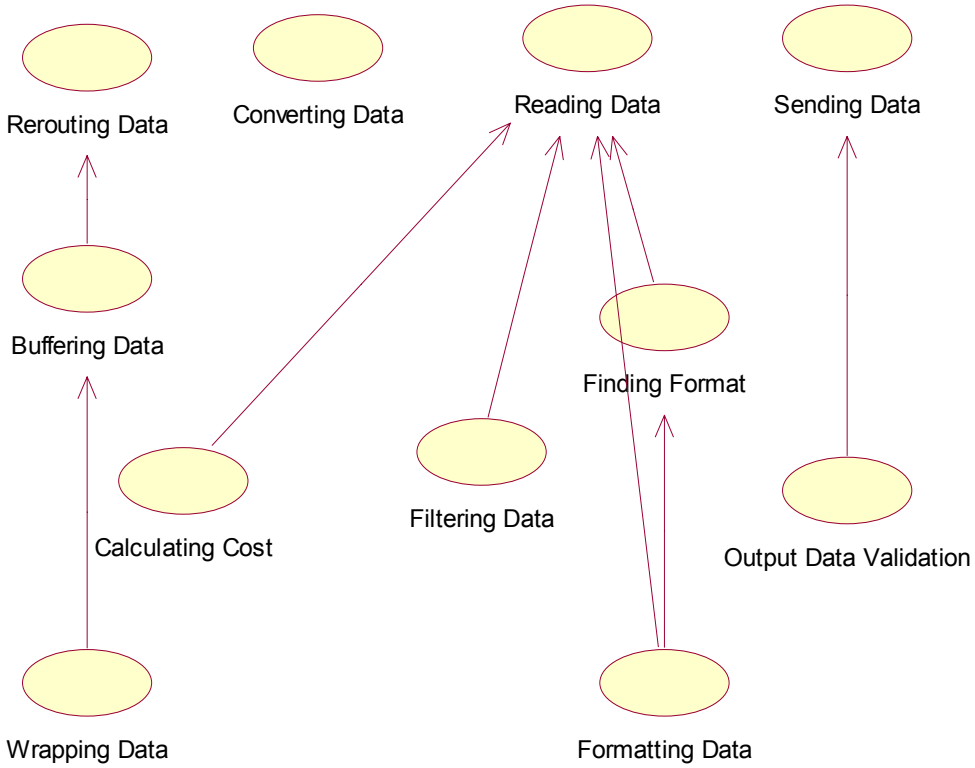


Figure 19. Processing data module use

1

Module 2: Logging Data

It covers the use cases that have the goal of store the information. The main use case Logging Information and the auxiliaries Logging Error Messages, Logging Input Data and Archiving Output Data are in this module.

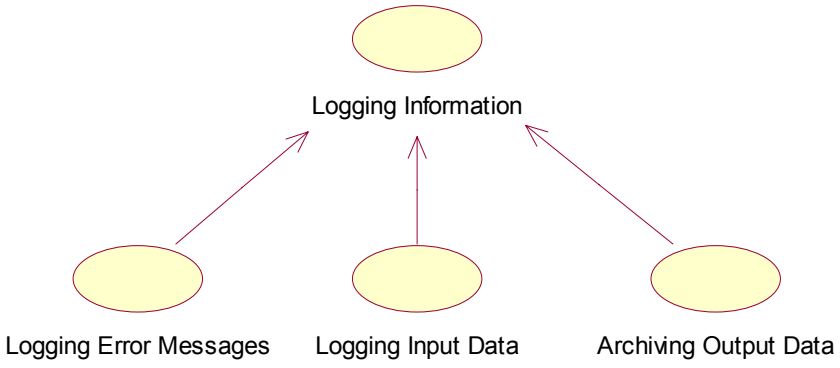


Figure 20. Logging data module use case diagram.

Module 3: Monitoring
Use cases to visualizing and controlling the available information are in this module. The main use case Monitoring and the auxiliaries Errors Control, Error Handling, History Manager, Monitoring Log, Monitoring Errors and Monitors Manager are in this module.

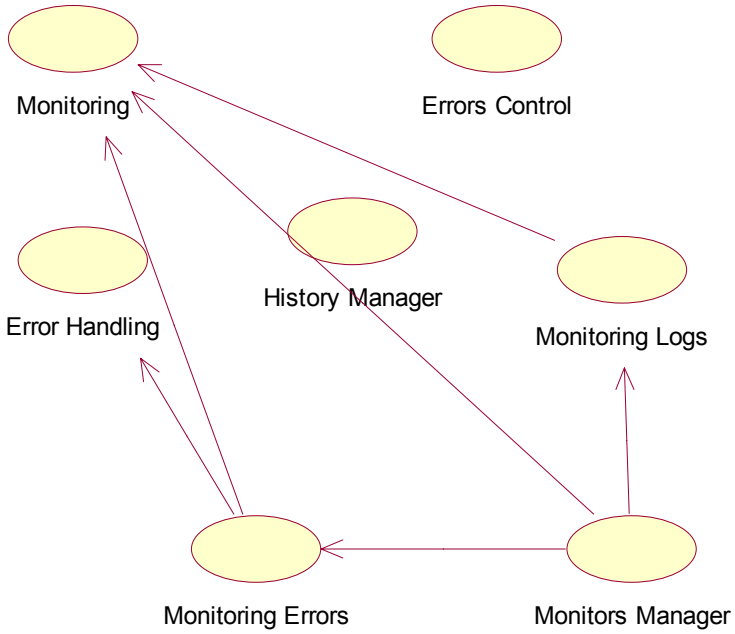


Figure 21. Monitoring module use case diagram.

Module 4: Measurements
This module has the use cases with the functionalities to execute the measurements in the system. This module has the main use case Measurements and the auxiliaries Measuring Input Data, Measuring Interpreted Data, Measuring Formatted Data, Measurement Costs, Generating Measurement Statistics, Measuring Output Data, Generating Error Statistics, Generating Measurement Reports, Measurement Manager.

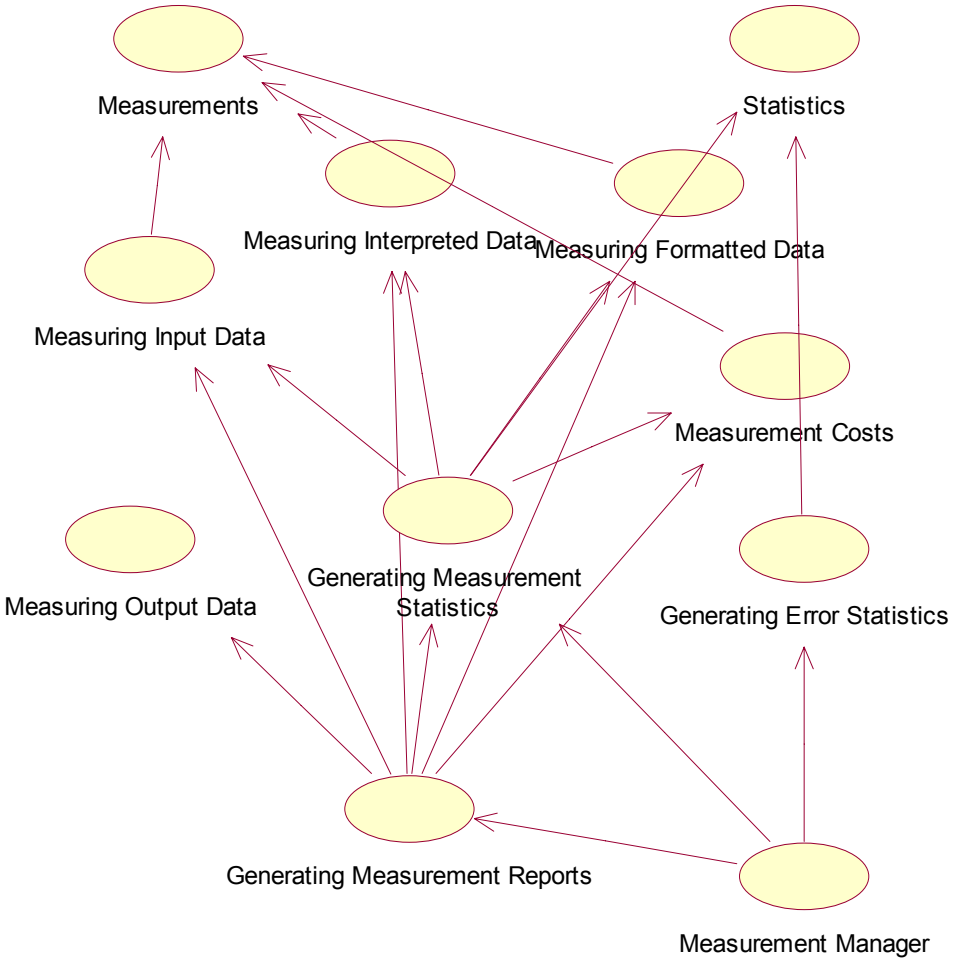


Figure 22. Measurements module use case diagram.

Module 5: Protocols and Formats
All protocols and formats definition use cases are in this module. This module has the main use cases Defining Formats and Defining Protocols, and the auxiliaries Defining Standard Format, Adding Interpreter and Adding Protocol.

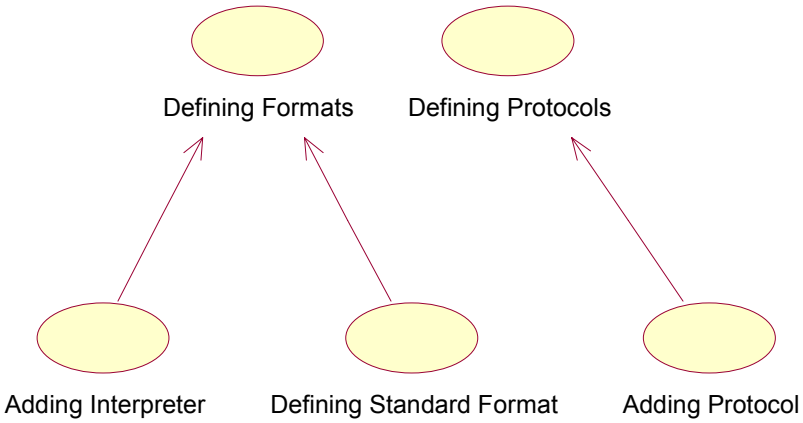


Figure 23. Protocols and formats module use case diagram.

3.7 Defining the Dependencies among the Use Cases

The use cases have dependencies among them. This represents the relationship existent in the modeling. The dependence means that one use case depends on another on to this execution. The eleven main use cases defined are not dependent on others, they can exist by themselves. The other use cases, which we called here auxiliaries, are dependent on them. Therefore, they are considered the principal use cases and they are so important to define the software development schedule. The relationships among the use cases in the same module are represented in the diagrams shown in the previous section. The relationships between use cases from different modules are represented in Figure 24, which has the dependencies between the modules.

In the *Message Exchange System* example, all the modules are analyzed together. However, system involving a large number of modules or many use cases in a module should be processed separated. Furthermore, if the modules don't have dependence among them, they also should be processed separated. In this last case, the pair wise comparison can be applied to each module, thereby priorities are obtained to each one. The ranking of priorities is analyzed for each module what can be more useful then comparing all together. In this way, the more important use cases of each module are easier identified.

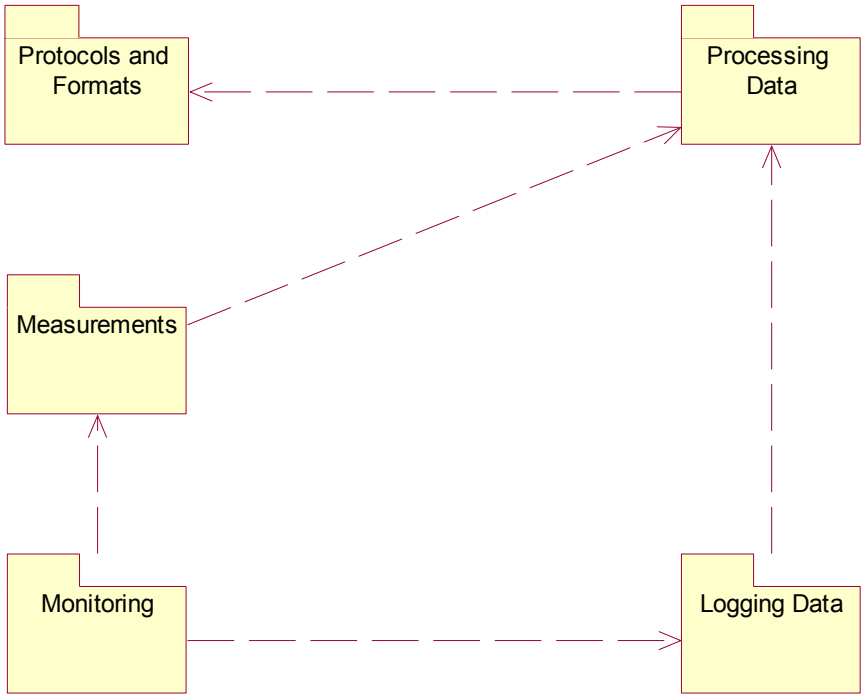


Figure 24. Dependencies among the modules.

3.8 Pair Wise Comparison of the Use Cases

The *Message Exchange System* example has 11 requirements at the first level of the hierarchy, therefore the matrix to compare those requirements is 11 x 11.
The values of the comparisons were given based on the idea to reduce the quantity of requirements to develop. The criterion is to have the main basic funcionalities in the system, while eliminating the least necessary requirements in implementing the system.

The first row and column have the use cases.
Making all the necessary judgements and reciprocals, the following matrix is obtained:

	Reading Data	Logging Information	Measurements	Errors Control	Statistics	Monitoring	Defining Formats	Converting Data	Rerouting Data	Sending Data	Defining Protocols
Reading Data	**1**	5	7	4	8	9	4	3	3	2	3/2
Logging Information	1/5	**1**	2	1/2	3	4	1/2	1/3	1/3	1/4	1/2
Measurements	1/7	1/2	**1**	1/2	3	4	1/2	1/4	1/4	1/5	1/3
Errors Control	1/4	2	2	**1**	4	5	2	1/2	1/6	1/7	1/2
Statistics	1/8	1/3	1/3	1/4	**1**	1/2	1/3	1/4	1/5	1/6	1/3
Monitoring	1/9	1/4	1/4	1/5	2	**1**	1/3	1/5	1/4	1/3	1/2
Defining Formats	1/4	2	2	1/2	3	3	**1**	1/2	1/3	1/4	1/2
Converting Data	1/3	3	4	2	4	5	2	**1**	1/2	1/3	3
Rerouting Data	1/3	3	4	3	6	4	3	2	**1**	1/2	4
Sending Data	1/2	4	5	3	7	3	4	3	2	**1**	3
Defining Protocols	2/3	2	3	2	3	2	2	1/3	1/4	1/3	**1**

Table 22. *Message Exchange System* comparison matrix.

For instance, *Reading Data* is considered an important requirement because it is responsible for receive the messages, the input data. Comparing *Reading Data* with the others requirements, the judgments take into account that this is a very important requirement.

The position in the matrix where *Reading Data* is compared with *Logging Information* has the value **5**, this means that R*eading Data* is considered **5** times more important then *Logging Information*, or strongly more important.

Comparing *Statistics* with *Error Control*, the last one is more important. The value inserted in this position is **1/4**. That means that *Error Control* is 4 times more important then *Statistics*. Comparing *Logging Information* with *Statistics*, the first one is more important. The value inserted in this position is **3**. That means that *Logging Information* is **3** times more important then *Statistics*.

All the judgments were done in this way and Table 22, that represents the matrix, is the result of the pairwise comparisons. The relative comparisons commented here are shown up in the matrix. Analyzing the values given in the matrix it is possible to observe which requirement is more important in each pairwise comparison to this example.

3.9 Calculating the Priority of Use Cases

Looking to the example of the *Message Exchange System*, the method is applied to the matrix in Table 22 previously defined with the judgments.
After the execution to the first level, the priority values obtained in the calculation are the following:

Order	Use Cases	Priorities
1	Reading Data	23
2	Sending Data	19
3	Rerouting Data	15
4	Converting Data	11
5	Defining Protocols	8
6	Error Control	6
7	Defining Formats	5
8	Logging information	5
9	Measurements	4
10	Monitoring	3
11	Statistics	2

Table 23. Priority values for level 1.

The first column indicates the order in the level, the second means the name of the use case and the last column brings the priority values.
The results presented in this section were collected from the tool developed to this approach. The use cases described for the system (Appendix A) were loaded into the system and the relative comparisons were also inputted into the system. The relative comparisons utilized are the same as the values defined in Table 22. After that, the process was executed in order to get the results.
Table 24 has the priorities for the other use cases. The first column indicates the order in the level and the second column indicates the level where the use case is located.

Order	Level	Use Cases	Priorities
1	2	Buffering Data	20
2	2	Measuring Formatted Data	15
3	2	Adding Protocol	12
4	2	Archiving Output Data	11
5	2	Measuring Output Data	9
6	2	Finding Format	7
7	2	Logging Input Data	7
8	2	Measuring Input Data	6
9	2	Filtering Data	5
10	2	Calculating Cost	5
11	2	Defining Standard Format	2
12	2	Adding Interpreter	2
1	3	Output Data Validation	26
2	3	History Manager	22
3	3	Formatting Data	21
4	3	Wrapping Data	20
5	3	Measurement Costs	11
1	4	Logging Error Messages	72
2	4	Measuring Interpreted Data	28
1	5	Generating Measurement Statistics	44
2	5	Monitoring Logs	31
3	5	Error Handling	25
1	6	Generating Measurement Reports	49
2	6	Generating Error Statistics	26
3	6	Monitoring Errors	25
1	7	Measurement Manager	63
2	7	Monitors Manager	37

Table 24. Priority values to the other levels.

3.10 Calculating the Consistency in Use Case Prioritization

The consistency ratio calculated to the *Message Exchange System* is 0.07. This consistency index was taken from the tool of this approach. The index correspond to the priority values shown in Table 23 and the judgments defined in Table 22. This value belongs to the first level. 0.07 means that the result is acceptable and the judgements are enough consistent. This rule was explained in section 2.4.3.3.3.
The other levels have 0.00 to the consistency ratio, because the values used to fill the matrix are calculated as explained in paragraph 2.4.3.3.3. So, there is no inconsistency in these values because they are deduced from the previous level.

Chapter 4

Assessment of the Prioritization Process

4.1 Introduction

In this chapter the assessment of the prioritization process is discussed. The results obtained with the *Message Exchange System* in the previous chapter are analyzed and commented. Also some actions which can interfere in the result are described.

4.2 Interpretation of the Acquired Results

Chart in Figure 25 shows the results obtained with the prioritization method. The values the priorities for each use case. These use cases belong to the first level.
The objective used in the criteria to apply the judgments was to produce a system to exchange messages with the minimum functions. We can see that the objective was caught up. The most important use cases in the ranking are *Reading Data*, *Sending Data* and *Rerouting Data*. They are fundamental to the exchange message software.

Reading Data is the requirement with the larger value. This requirement represents one of the main activities in the system, so this part of the result reflects the goal of the prioritization. The interpretation could be that this requirement should be implemented. The responsible person or time by the project should analyze the result and make decisions about it.

Statistics and *Monitoring* are the requirements with the lowest priority. This means that these requirements are not so important to the system. The responsible by the project should decide if these requirements should or not be developed. If the idea is to reduce the number of requirements, Statistic and Monitoring must to be analyzed in this way. One of them or both can be skipped. However, the priorities can also be used to decide the order of the development. For instance, the use cases with higher priorities will be developed in the first phase of development and the use cases with lower priorities will be developed in the next phase of development. The method shows the priorities like options to the development of the project.

Based on this rank, the project manager can plan the strategies to the software development schedule. Using the information provided, managers can make decision about what should be implemented or not and when. The closer the priority values, more carefully they should be analyzed.

The other levels bring information about the use cases that are dependent on the most important ones. The use cases dependent on some use case that may be skipped in the development probably will be also skipped, because their functionalities are dependent on the others. Or they will be developed in the same iteration of the use cases which there are dependent. Their priorities helps to define the order of development.

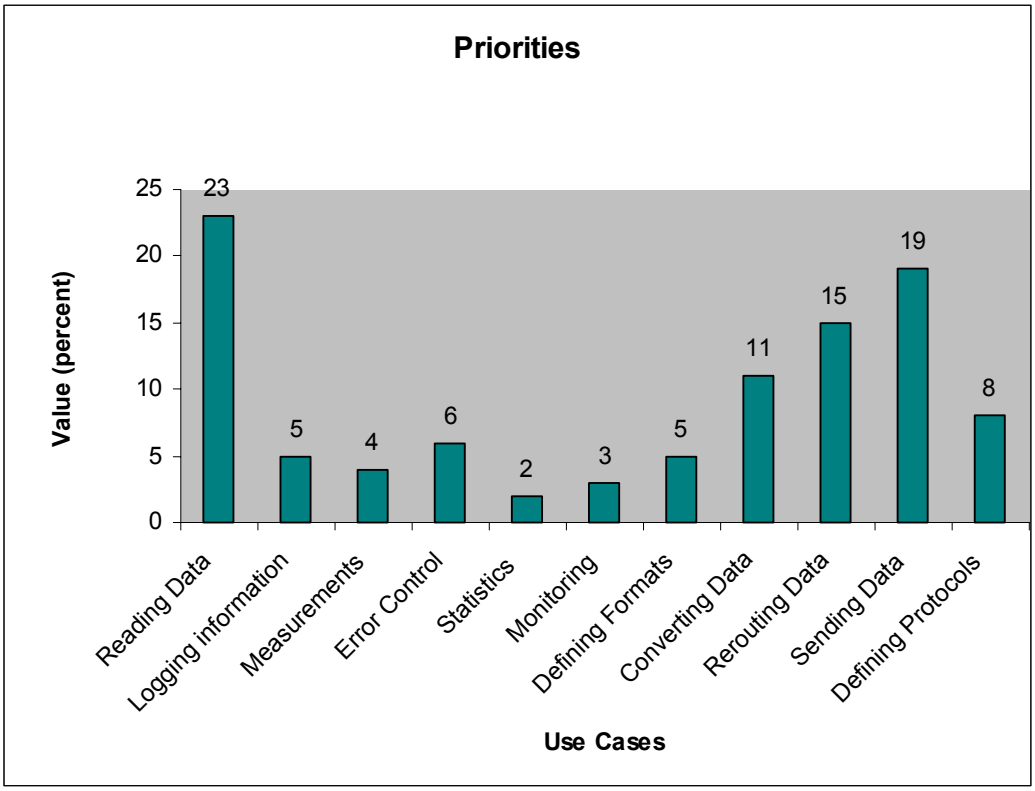


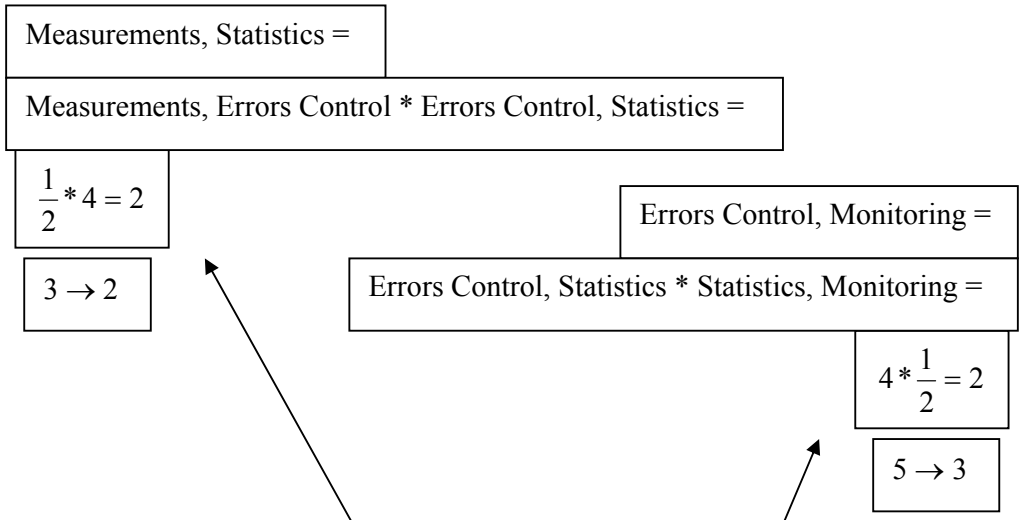
Figure 25. Chart with the priority values.

4.3 Revising Results based on the Consistency Value

The consistency value obtained in the *Message Exchange System* is equals to 0.07. This value is acceptable, as explained in section 2.4.3.3.3 because it is less then 0.10. This consistency index indicates that the judgments are consistent. But to improve the consistency value, the recommendations explained in section 2.4.3.3.4 can be used. These rules will be used here to demonstrate the result of the changes in the relative comparisons and the priorities.

Firstly, we assume that the first diagonal above the main diagonal is the parameter, because we need some base values to do the calculation. The inconsistencies can be anywhere in the matrix. If it is really necessary to reduce the inconsistencies the decision maker should firstly review the judgements, analyzing the alternatives again.

After that, the steps described in this section can be executed. This section only describes some actions to try to find inconsistencies. If the consistency index is not improved changing the values based on the first diagonal chosen, the same steps can be executed with other diagonal. Then we can try making the multiplication and compare the results. Table 25 shows an experience where two values are tested and changed.



	Reading Data	Logging Information	Measurements	Errors Control	Statistics	Monitoring	Defining Formats	Converting Data	Rerouting Data	Sending Data	Defining Protocols
Reading Data	**1**	5	7	4	8	9	4	3	3	2	3/2
Logging Information	1/5	**1**	2	1/2	3	4	1/2	1/3	1/3	1/4	1/2
Measurements	1/7	1/2	**1**	1/2	2	4	1/2	1/4	1/4	1/5	1/3
Errors Control	1/4	2	2	**1**	4	3	2	1/2	1/6	1/7	1/2
Statistics	1/8	1/3	1/3	1/2	**1**	1/2	1/3	1/4	1/5	1/6	1/3
Monitoring	1/9	1/4	1/4	1/3	2	**1**	1/3	1/5	1/4	1/3	1/2
Defining Formats	1/4	2	2	1/2	3	3	**1**	1/2	1/3	1/4	1/2
Converting Data	1/3	3	4	2	4	5	2	**1**	1/2	1/3	3
Rerouting Data	1/3	3	4	3	6	4	3	2	**1**	1/2	2
Sending Data	1/2	4	5	3	7	3	4	3	2	**1**	3
Defining Protocols	2/3	2	3	2	3	2	2	1/3	1/4	1/3	**1**

Table 25. *Message Exchange System* comparison matrix with changes.

The first case, (Measurements, Statistics), had the value 3, but in the calculation the value 2 was obtained. So, we changed the value.
In the second case, (Errors Control, Monitoring), had the value 5, but the calculation resulted in the value 2. But we decided to use the value 3. Any way the consistency value will be improved because 3 is better then 5.

The priority values didn't change. They are the same, because the changes didn't represent a big change. For instance, if we changed the value 3 to 1/3, it would be a change with more influence in the result because the comparison would be inverted.

The consistency value in the example is 0.07 and with this two modifications the value is 0.065. The consistency was improved but the priorities didn't change. This means that we can have different values, but nearly the same and the result may be the same. This represents a margin of values to the customer to apply the pair wise comparisons. There is no only one right answer for each judgment.

4.4 Dealing with Incomplete Data

It is better if all the comparisons can be provided, but it is also possible apply to the method with some values missing [HAR87a] [HAR87b] describe some techniques to reduce the number of pairwise comparisons that the decision maker must make during analysis of a large case. In this section, a technique to reduce the amount of work needed to compare elements is discussed.
As discussed in paragraph 2.4.3.1.4, $n \cdot (n-1)/2$ questions must be answered to fill the entire matrix. However, $n-1$ questions could be requested for filling the first row, for instance. But the redundancy in questioning is essential to obtain reasonable estimates of priorities. For the missing matrix entry aij, their values are approximated by the ratio of the weight wi/wj.
For example, this matrix has entry (1, 3) missing:

$$C = \begin{pmatrix} 1 & 2 & w1/w3 \\ 1/2 & 1 & 2 \\ w3/w1 & 1/2 & 1 \end{pmatrix}$$

Computing the value of Cw, the following vector is obtained:

$$\begin{pmatrix} 1 & 2 & w1/w3 \\ 1/2 & 1 & 2 \\ w3/w1 & 1/2 & 1 \end{pmatrix} . \begin{pmatrix} w1 \\ w2 \\ w3 \end{pmatrix} = \begin{pmatrix} 2w1 + 2w2 \\ 1/2w1 + w2 + 2w3 \\ 1/2w2 + 2w3 \end{pmatrix}$$

The vector could be obtained from multiplying the following matrix A by w:

$$A = \begin{pmatrix} 2 & 2 & 0 \\ 1/2 & 1 & 2 \\ 0 & 1/2 & 2 \end{pmatrix}$$

This means that Aw = Cw.

Analyzing matrix A, 0 is set in the matrix when the value is not filled and 1 is added to the main diagonal for each missing entry in a row.

Applying the computational procedure to the matrix A makes:

$$w = \begin{pmatrix} 0.46 \\ 0.33 \\ 0.21 \end{pmatrix}$$

The value of λmax = 3,1. How to calculate λmax is described in Section x, but in this case it means that the result is valid, as closer the value of λmax to the number of elements is better. In this example, the value is 3.

The incomplete comparison method allows reducing the effort.
But the priorities will be more precise as more values are inserted in the matrix.
To assure a reasonable result, in this approach it is assumed that at least one comparison value per row should be informed, because in this way there is at least one comparison for each requirement.

Using the *Message Exchange System* example with the matrix of comparisons defined in section 3.8, next table shows the matrix with two missing values.

	Reading Data	**Logging Information**	**Measurements**	**Errors Control**	**Statistics**	**Monitoring**	**Defining Formats**	**Converting Data**	**Rerouting Data**	**Sending Data**	**Defining Protocols**
Reading Data	**1**	5	7	4	8	9	4	3	3	2	3/2
Logging Information	1/5	**1**	2	1/2	3	4	1/2	1/3	1/3	1/4	1/2
Measurements	1/7	1/2	**2**	1/2		4	1/2	1/4	1/4	1/5	1/3
Errors Control	¼	2	2	**1**	4	5	2	1/2	1/6	1/7	1/2
Statistics	1/8	1/3		1/4	**2**	1/2	1/3	1/4	1/5	1/6	1/3
Monitoring	1/9	1/4	1/4	1/5	2	**1**	1/3	1/5	1/4	1/3	1/2
Defining Formats	1/4	2	2	1/2	3	3	**2**		1/3	1/4	1/2
Converting Data	1/3	3	4	2	4	5		**2**	1/2	1/3	3
Rerouting Data	1/3	3	4	3	6	4	3	2	**1**	1/2	4
Sending Data	1/2	4	5	3	7	3	4	3	2	**1**	3
Defining Protocols	2/3	2	3	2	3	2	2	1/3	1/4	1/3	**1**

Table 26. *Message Exchange System* comparison matrix with missing values.

It was supposed that these pairwise comparisons are missing:

- *Measurements, Statistics* and
- *Defining Formats*, *Converting Data*

The method described to deal with missing values was applied to the matrix, so the rows where there is the value 0 have the value 2 in the main diagonal.
Next table shows the priority values to the requirements using both comparisons matrices. The first column with priority values was obtained from the calculations in the matrix represented in Table 22, that is the complete matrix, with all the values.
The second column with priority values was obtained from the calculations in the matrix represented in Table 26, where there are the missing values.

Order	Requirement	Priorities Table 22	Priorities Table 26
1	Reading Data	23	23
2	Sending Data	19	19
3	Rerouting Data	15	15
4	Converting Data	11	10
5	Defining Protocols	8	8
6	Error Control	6	6
7	Defining Formats	5	6
8	Logging information	5	5
9	Measurements	4	3
10	Monitoring	3	3
11	Statistics	2	2

Table 27. Comparing priority values.

Comparing the results, there is no large difference in the values. The percentage to the requirements whose values in the relative comparisons were changed has some variance in the priority. But the order is the same. The result is not so affected.

The consistency ratio (CR) for the matrix in Table 26 is 0.069, this means that the judgment values are consistent. CR was calculated as described in 2.1.2.3.

Chapter 5

Designing a Tool for Prioritizing Requirements

5.1 Introduction

In this chapter the tool developed to prioritize requirements in this project is described. The system design was specified in terms of a UML Class Diagram.
The goal of the tool is to calculate priority values to the desired elements. In this case the elements are use cases representing the requirements.
The core of the tool can calculate priorities as described in chapter 2 to requirements and also to problems and solutions. The graphic interface developed to interact with the core of the tool works with Rational Rose models. Use case models are the input to the system. The use case diagram is parsed and transformed into objects that are managed to extract information and store new information by the system.
The system source code has been documented using Java's javadoc utility. All the classes in the core have been documented. Javadoc is the tool from Sun Microsystems for generating API documentation in HTML format from doc comments in source code.

5.2 The Tool Architecture

The Prioritization of Requirements application was written in Java. Java was selected for the main reason that the tool would be integrated with other tools. The other tools have already being developed in Java, so this was the best choice. Moreover, since Java [JAVa] is an interpreted language, it is portable to a wide range of hardware platforms. The version used is JDK 1.4 [JAVb].
The Java platform is widely used nowadays. Java allows building object-oriented programs as to corporative environment as to small applications.

The Analysis and Design of the system were developed using Rational Rose [RAT] version 2001 and 98i. For developing, Eclipse [ECL] platform, which is delivered has an integrated development environment (IDE), was used.
Jgraph [JGR], which is a library, was used to build the graphs. This library helps to generate the graphic representation of the objects. JGraph is an open-source graph component available for Java. It works in any Java application that uses *Swing*. This software provides a graphical view of the business or domain specific data.

The Swing package is part of the Java Foundation Classes (JFC) in the Java platform. The JFC encompasses a group of features to help people build GUIs; Swing provides all the components from buttons to split panes and tables.
Figure 26 shows the tool architecture. `PriorityApplication` is the main frame, and this frame has communication with the other frames and the classes manager. The other frames are `GraphicInput`, `MatrixFrame`, `ResultFrame`, `ConsistencyFrame` and `GraphicResult`. The classes manager are `PriorityManager`, `DataManager`, `MatrixManager` and `ConsistencyManager`.

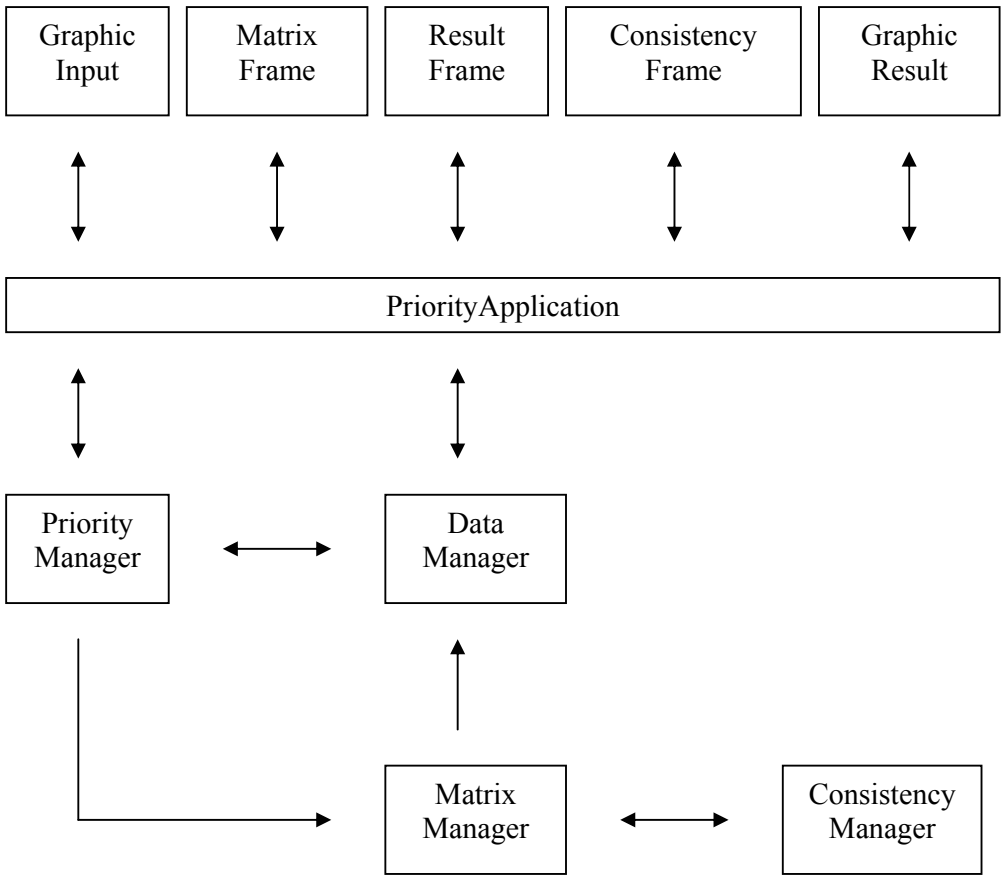


Figure 26. The tool architecture.

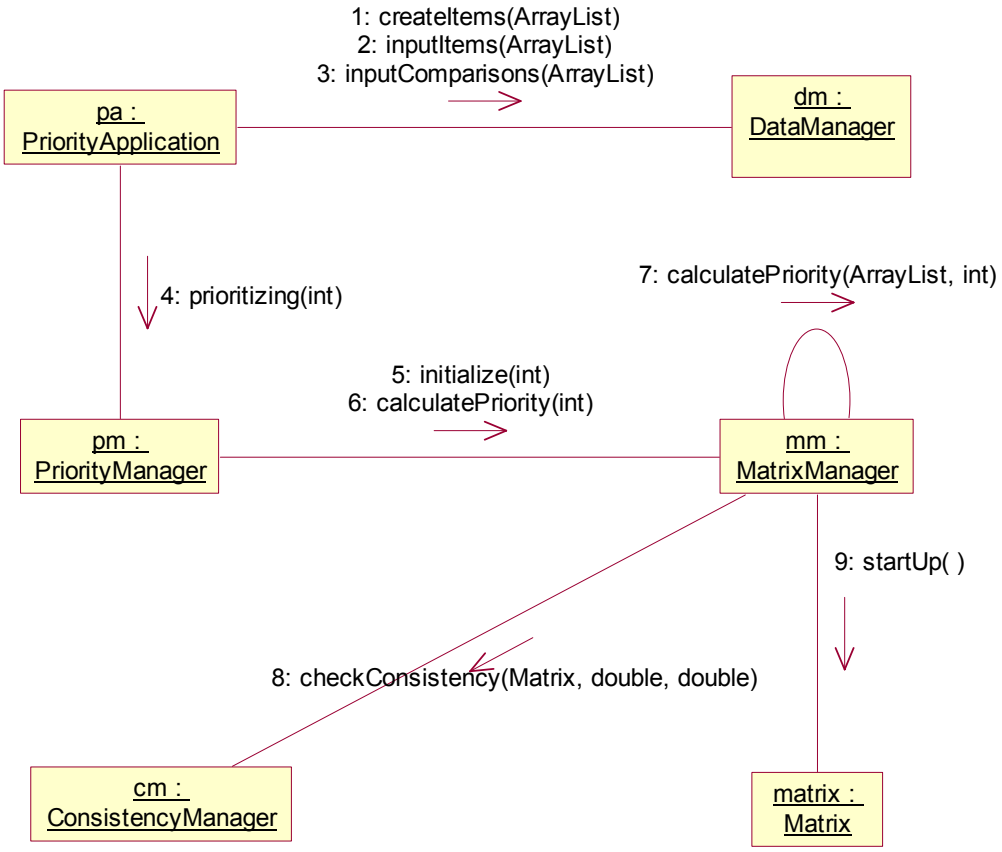


Figure 27. Collaboration Diagram of the core of the Tool.

The collaboration diagram in Figure 27 shows the main part of the tool, where the method is executed. The application starts in `PriorityApplication` class, this class receive the necessary information (use case and relative comparisons) from the other frames. This class interacts with the classes manager in order to control de data and the method execution. Firstly, `PriorityApplication` interact with `DataManager`, when items, dependencies and comparisons are created in the system. After that, the prioritization process can be started. `PriorityApplication` calls `PriorityManager` to start the process. `PriorityManager` interacts with `MatrixManager` to execute the calculation. `MatrixManager` controls the priority calculation process. `MatrixManager` asks Matrix to calculate the values. After the definition of the values, `MatrixManager` interacts with `ConsistencyManager` to obtain the consistency index. This is briefly the system flow. The classes are explained with more details in the next sections.

5.3 Design

The design of the Tool was broken up into four distinct modules: Priority, Matrix, Collection and the Graphical User Interface (GUI). The modules were divided taking into account the main aim of the classes. Each module is discussed in details in the next sections.
Design diagrams describing the interaction of these classes is shown in Figures 28 through 34. These diagrams were built using UML.

5.3.1 Priority Module

Priority Module contains the classes responsible for control de prioritization.
The `PriorityManagerInterface` interface allows to get the priority value of the requirements in the system. Any class can implement this interface to be able to access the priority values.

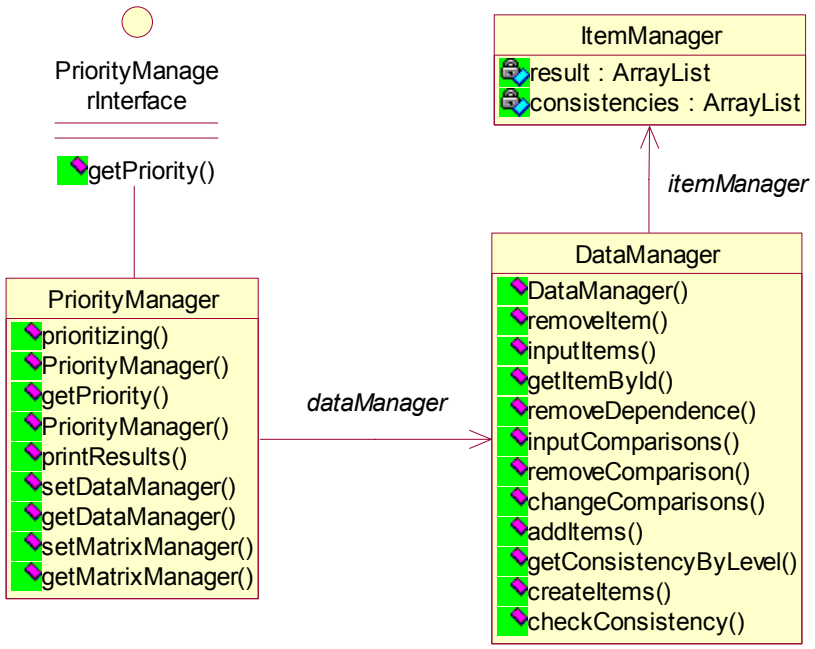


Figure 28. Class diagram of Priority Module.

An interface is a reference type whose members are constants and abstract methods. This type has no implementation, but otherwise unrelated classes can implement it by providing implementations for its abstract methods. Programs can use interfaces to make it unnecessary for related classes to share a common abstract superclass or to add methods to Object.

`DataManager` class controls everything related to the elements in the tool. The elements mean all the objects created from the input data, for instance the requirements or use cases. Each element is represented by an Item in the system. Item is described later in this section. All the operations of insertion, deletion and modification of Items are requested to this class. Besides the Items also the relative comparisons collected in the matrix are managed by the `DataManager`.

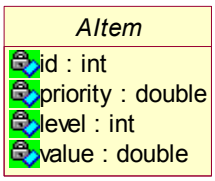


Figure 29. Abstract class *`AItem`*.

Every element inserted in the system is represented by an Item object.
`Item` class extends the abstract class *`AItem`*.
`AItem` has the necessary information about each element as id, to identify the item; priority, the calculated value; level, the level of the item in the hierarchy and value, the value of the item used in the calculation. This value can be the relative comparison or deduced from the dependence relationship.

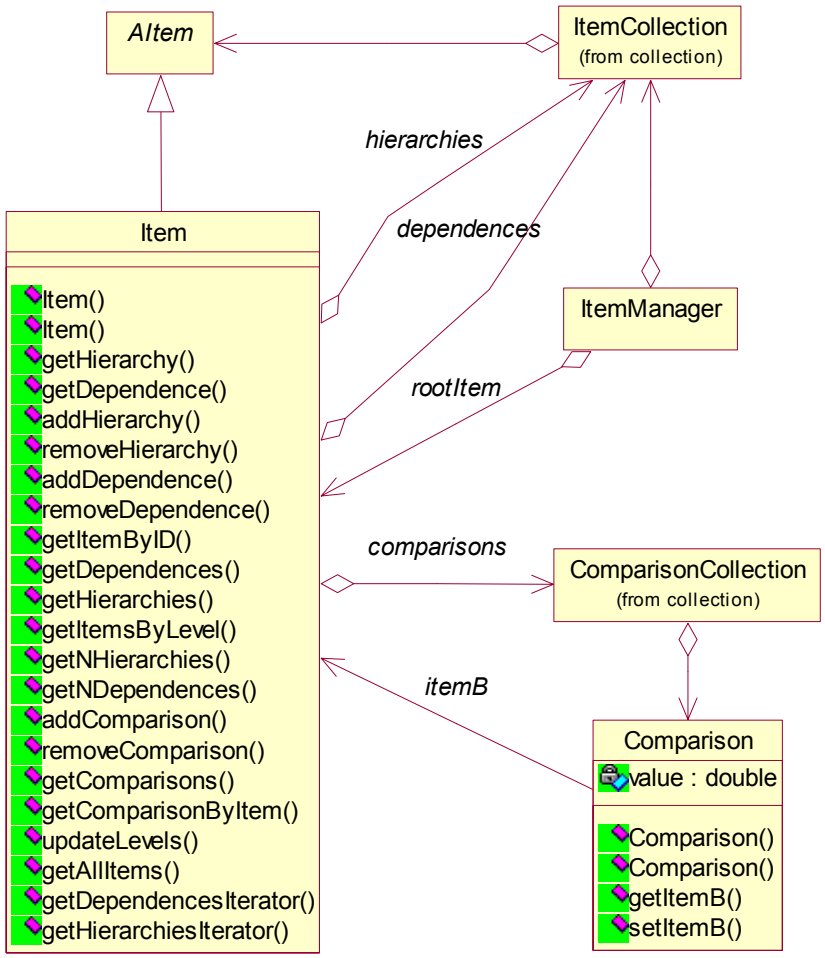


Figure 30. Class diagram of Priority Module, Item representation.

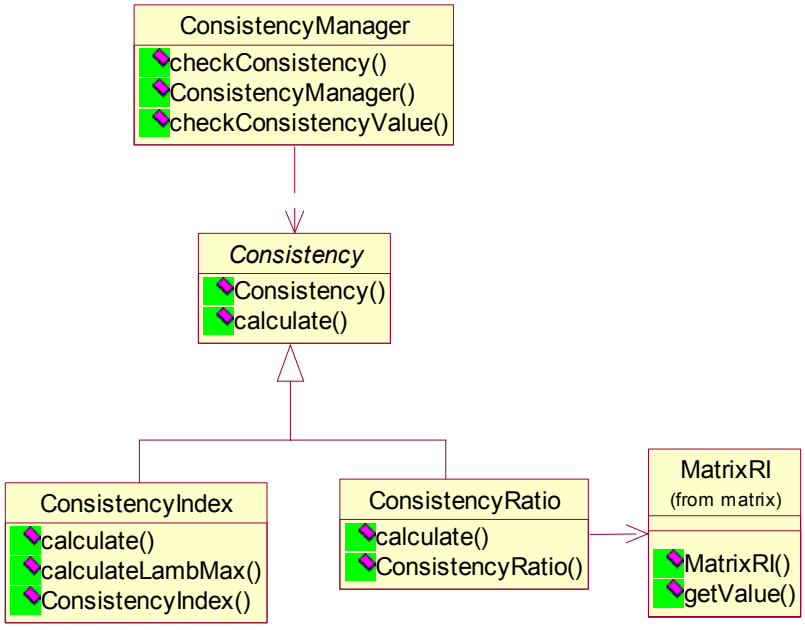


Figure 31. Class Diagram of Priory Module – Consistency Part.

These classes belong to the Composite Pattern.
Composite composes objects into tree structures to represent part-whole hierarchies.
It lets clients treat individual objects and compositions of objects uniformly [GAM95].
Items are organized in a tree. The hierarchy is defined by the relationships between the elements. Each Item has a collection with the Items that are its dependent. This way it is possible to cover all the elements in the tree. The collection classes are explained in section 5.3.3.
The calculation of the inconsistencies is also in this module.

5.3.2 Matrix Module

This module consists in the representation of the data in matrices.
`MatrixManager` class controls the matrices involved in the process and also the process of calculation.

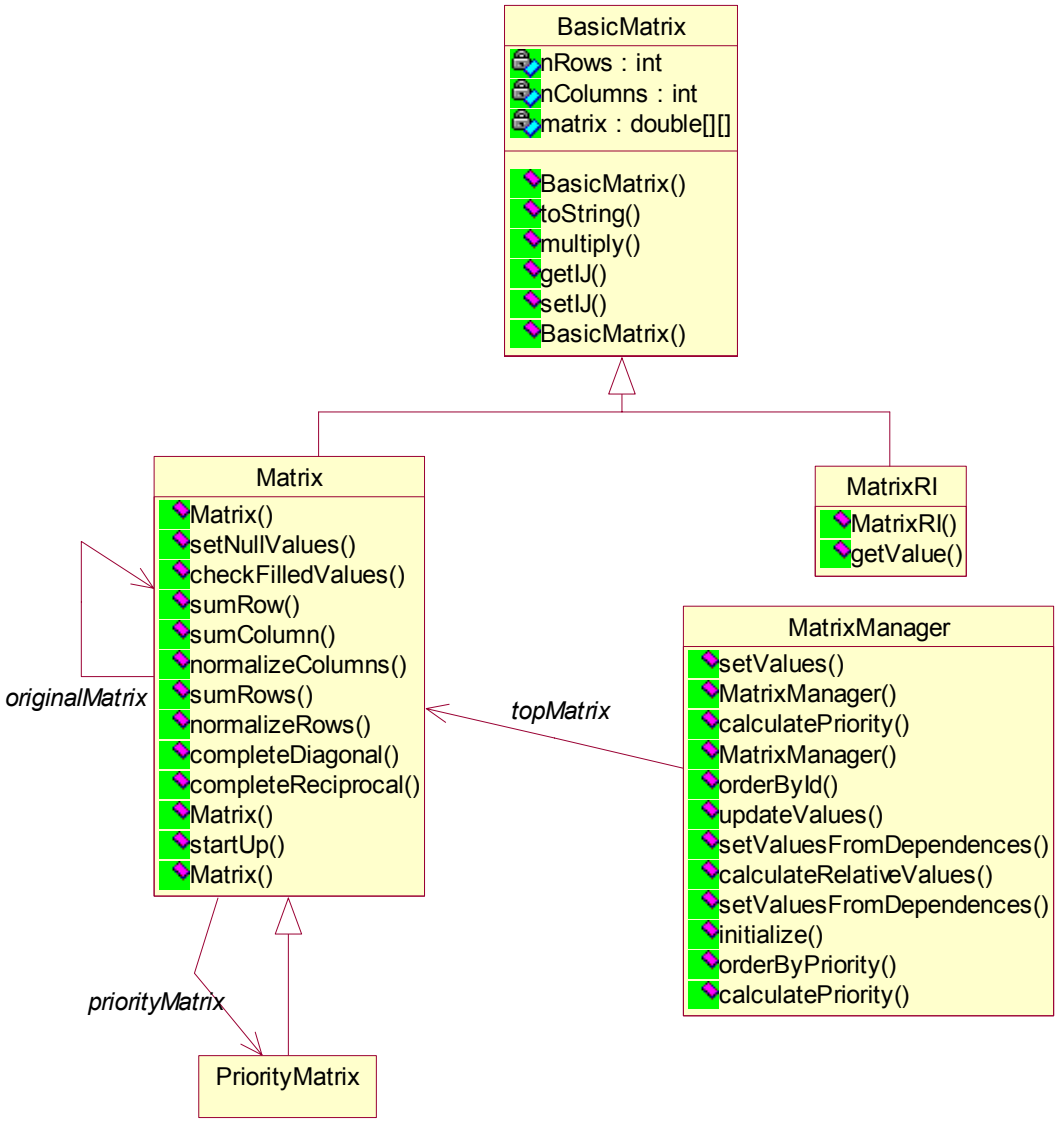


Figure 32. Class Diagram of Matrix Module.

`MatrixRI` class has the pre calculated values to the random indices (RI). These values were defined as described in chapter 3.
There are values defined to matrix of order between 3 and 15. Matrix of order 1 and 2 has the RI equals zero. These values were enough to the experiments, because the experiments have less then fifteen requirements each one, this means that is not necessary matrices with order larger than 15.
It is possible to calculate random indices to matrices of other sizes.
A class called `RIvalue` was implemented to be able to define random indices. The desired order of the matrix is informed and the process is executed to calculate the value.

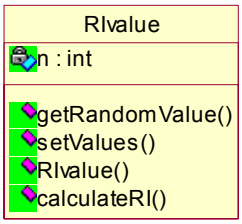


Figure 33. `RIvalue` class.

5.3.3 Collection Module

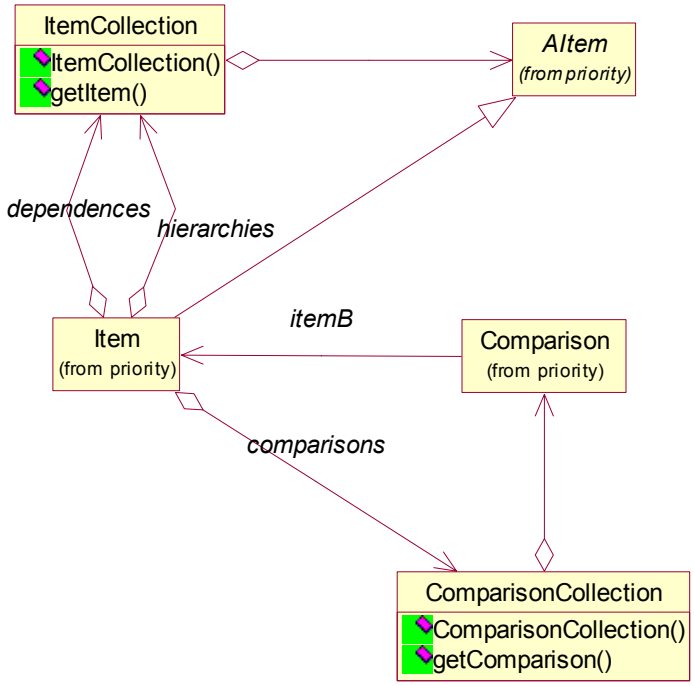


Figure 34. Collection Module.

Object specific wrapper classes were implemented. This was done to improve code legibility, since it makes it clearer as to what the nature of the affected variable is.
It also reduces the complexity of the class diagrams. The aggregation relationships between a class and other classes are replaced with an aggregation 1 to 1 relationship between the specific collection class and the other classes and another one between the class and its collection.

The collection classes extend the `ArrayList` class. `ArrayList` is a resizable-array implementation of the `List` interface. `List` is an ordered collection (also known as a sequence). The user of this interface has precise control over where in the list each element is inserted. The user can access elements by their integer index (position in the list), and search for elements in the list.
There are two collection classes: `ItemCollection` and `ComparisonCollection`.
`ItemCollection` as its names says is a collection of items represented by the abstract class *`AItem`*.
`ComparisonCollection` is a collection of comparisons.

5.4 Integrated Tools

This prioritization tool is integrated with other tool. The tool aims at supporting software developers in the early phases of software development projects that may evolve or not. Early phases in software development are the requirements engineering phase to identify the requirements of a system, the analysis phases to identify use-cases of the system, and the architecture phase to identify packages of the system.
The tool is a software development environment framework in which modules that support developers can be inserted. It is based on graph theory, which also provides the formal foundation for the tool. Modules communicate among each other by using graphs. To integrate in existing software development environments, the tool is designed to connect to other development environments.
At this moment the case tool Rational Rose is supported.

The tool supports the approach described in [GLA03] that uses problem patterns for mapping requirements to technical problems. That approach uses synthesis with problem patterns. A pattern is a description of a solution to a common problem in software. The method is the same to develop the software architecture, the difference is that the problems are identified in the problem pattern base. A problem pattern can be more efficiently understood and more effectively transformed. The problem patterns are searched in a database. The database stores the previously defined problem patterns categorized with respect to certain domain classification schema.
A Domain Specific Design Pattern is a set of objects and components that form a highly encapsulated, cohesive partition with clear boundaries, which can also be used in a specific software domain.
Domain specific design patterns helps to address the problems with general design patterns. A domain specific design pattern would be a pattern that is in some way optimal for that particular domain. That is, they have a clearly defined scope in which they solve a problem. They also provide the designer with well-defined partitions that suit the specific domain [POR98] [GUS02].

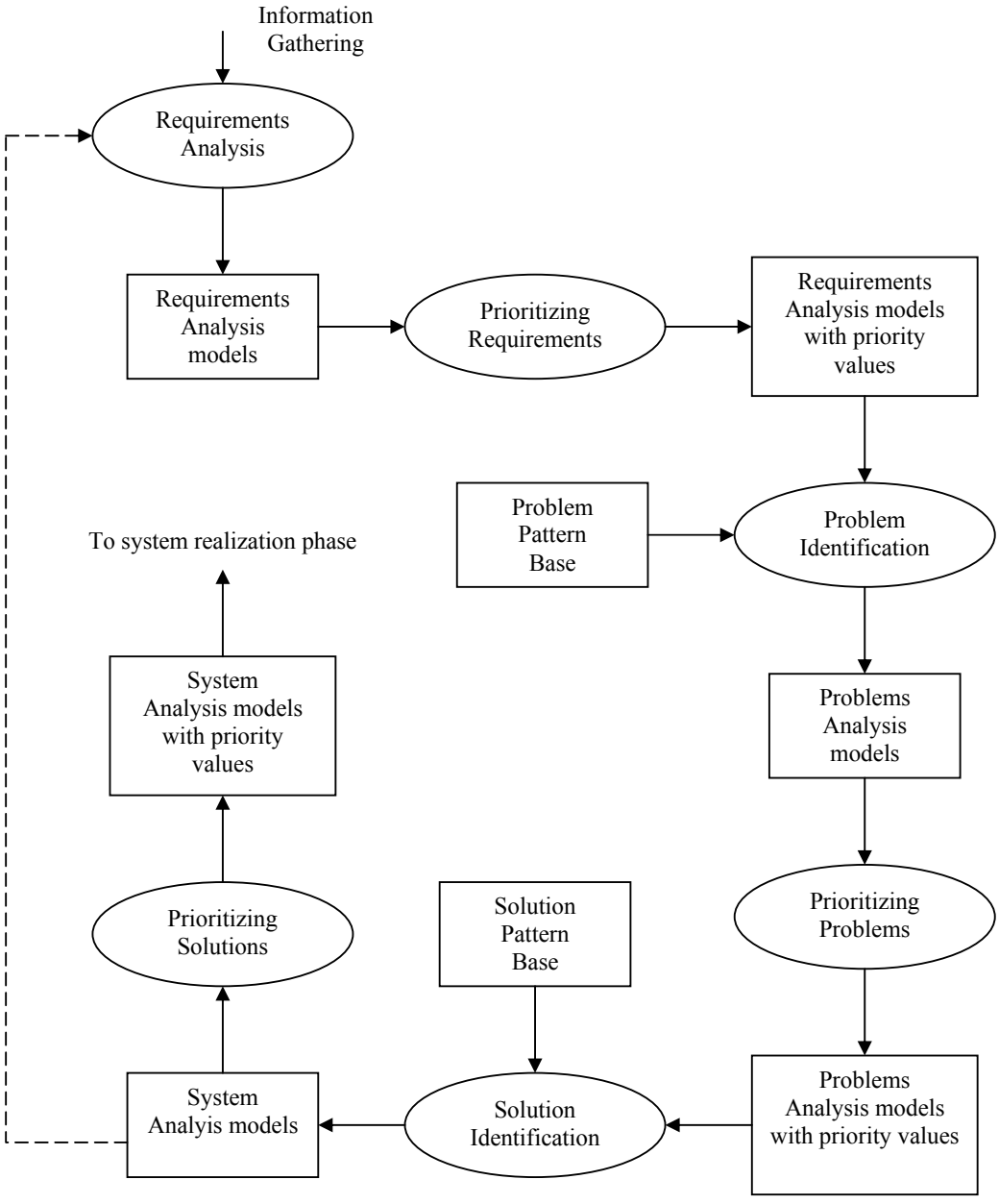


Figure 35. Transformation process with prioritization.

Figure 35 has the synthesis transformation process, which is used to transform requirements analysis models to system analysis model with problem patterns. The models and process are represented as rectangles and ellipses, respectively. The process of prioritization is included in this transformation process to indicate where it can be applied. It starts with the *Information Gathering*. The first process is the *Requirement Analysis* and the *Requirements Analysis* models are prioritized. The prioritized requirements are processed by the *Problem Identification* that searches for

the problem patterns in the *Problem Pattern Base*. The *Problem Analysis Models* are prioritized and then processed by the *Solution Identification*. The solutions are searched in the *Solution Pattern Base*. After that, *System Analysis Models* are prioritized to finally go to the system realization phase. Additionally, if in the *System Analysis Models* new requirements are defined they go to the first phase, *Requirement Analysis*.

5.5 Graphical User Interface (GUI)

The purpose of the Graphical User Interface (GUI) is to facilitate the interaction of the user with the tool. Moreover the visualization of the data and the results is better to the user.
The GUI developed to this approach is simple and brief, but it brings the essential necessities. Users have the ability to input the requirements, control the prioritization and view the results.
The user interface was developed in the eclipse platform. The graphical view of the information is built with the JGraph [JGR] library.
The description of the user interface is in Appendix C.

Chapter 6

Conclusions

6.1 Related Works

The interest in the requirements engineering is growing in the software industry. Requirements engineering deals with the study of software requirements, so as to be able to accurately define what is to be built. This is the basis for the quality of the development. Also, there is great interest in the efficient definition of requirements. This includes the problem statement discussed by this thesis. This means that the set of requirements determined for a project must meet the need of this project, but also pay attention to time and budget constraints. So, an interesting problem in this area is selecting a requirements-choosing methodology, so that the resulting set fulfils the project constraints. Some research has been carried out in this area. Generally, the most common methodology is to select the most important requirements, without any further analysis. For example, a customer will usually tell the development team, which are the requirements with the highest priority by just signaling them out. Studies have determined the existence of some methodologies, involving scales and calculations. Papers related to this topic are detailed in the following paragraphs.

[WIE99]says that customers and developers must collaborate on requirement prioritization. Developers do not always know which requirements are most important to the customer, and the customer cannot judge the cost and technical difficulty associated with specific requirements. This approach considers the relative benefit, penalty, cost and risk when defining the priority of the requirements. Benefits indicate alignment with the product's business requirements. Penalty estimates the relative penalty the customer or business will suffer if the feature is not included. Cost represents the cost of implementing each feature. The last factor is the risk associated with each feature. These values are combined to give the priority.

In [RYA97] and [KAR97], AHP is adopted as the basis for a systematic methodology for prioritizing requirements. This approach estimates both the relative cost of implementing each candidate requirement and the relative value of this requirement to the eventual customer. The relative cost and value of each requirement are analyzed together.

[PAR99] evaluate the difficulty and importance of each requirement. Stakeholders assign each item a difficulty and importance for which summary statistics are generated and used to classify them into a relative prioritization. The votes, given by the stakeholder, are then summarized according to some agreed upon policy (such as majority rules, average value, median) and then a value is assigned to the requirement. The requirements are grouped with respect to some model based on the voting.

Example groupings are “high, medium, low” or “priority I, II, III”, which are defined according with the policy utilized.

6.2 Conclusions

In general, industrial software development projects have to cope with many requirements. However, it is difficult to deal with so many requirements at the same time, and therefore, software engineers have to - in some way - select and/or order requirements. Moreover, systems developed to be released by versions require the determination of the order of development of the requirements. Constraints like time and budget are commonly present in real projects and the coordination of the project to attend these necessities is very important. In most projects, the time constraint has the greatest influence. Effective prioritization of requirements can help in dealing with these problems. The problem of selecting the subset of customer requirements and still building a system that satisfies the main needs, can be solved with the prioritization of the requirements. In this thesis we presented a method based on the decision making process AHP to deal with prioritization of requirements.

This prioritization of requirements approach consists in a method that gives priority values to the requirements, in order to aid system management and software quality. Prioritization is essential for managing requirements. The process of prioritization will help reduce rework and schedule problems in projects. The need to prioritize increases with the number of requirements. Therefore, the prioritization process is fundamental in the planning of the system, making the project more reliable.

The priority values help the system analysts, developers and customers make decisions about the requirements. Based on the values given by the approach, requirements can be skipped or developed later. Knowing the rank of the requirements, the plan of the releases can be done. It is possible to plan by knowing which functions are critical and how they can be distributed. The closer the final priority values are to each other, the more careful the evaluation should be. Then, the schedule can be defined, adding the activities in the indicated order over successive releases. The priority information is very important when developing strategies and in consequence, works in benefit of the software quality, fulfilling the stakeholder's expectations and constraints.

Prioritization of requirements requires some effort and work from the customer, but it brings benefits to the project and in consequence to the customer. The resulting reduction in effort will be considerably more than the effort expended establishing priorities.

The consistency index of the results presented by this method helps in the analysis of the results that were obtained. Analysts, developer and customers, that executed the activity of making the judgements about the requirements, may have the notion about the consistency of relative comparisons done. This index allows the correction of possible judgement mistakes, producing more reliable results. This can require more effort, as it involves backtracking to the pair wise comparisons, but it compensates by producing a better outcome.

6.3 Future Work

This section presents the possibilities of new studies in the area.

The process of prioritization can be improved in the case of the software architecture methods based on synthesis. This kind of method is centered in the Problem Domain and Solution Domain. The elements or knowledge in the Domain Knowledge can be evaluated in order to observe the importance of the different kinds of knowledge when they are present in the same project. For instance, if the system project will work with the domain knowledge of database, graphic interface (GUI) and network management, the most important knowledge in this set can be indicated. Different projects should be evaluated to check if the same behavior is observed. The observation could be used to develop the possible rules about the relations between the different knowledge domains. Also, different kinds of systems should be analyzed because the behavior may follow some rule based on the main goal of the system. The rules could be values attributed to each specific knowledge depending on its importance against the others. The values could be used as weights in the prioritization process. So, when it is necessary to prioritize problem or solution domains, these values should be used to facilitate the decision making process. The values, or weights, should be stored together with the respective knowledge to be available when needed. This idea could be also applied to the methodologies using domain specific design patterns, because they have the same process of synthesis but the knowledge is based on patterns.

Another issue where this can also be applied is to methods based on synthesis taking into account the software development phases. The same activity proposed of observing the behavior in the different domains could be executed in the different phases of software development. In general, the software development phases are: Requirements, Analysis, Design, Implementation and Test. In the Requirements phase, the problems have not yet been identified. Moreover, in the Test phase, the solution domains have already been identified and applied. So, the Analysis, Design and Implementation phases could be evaluated. This means to evaluate if each domain knowledge has a different importance depending on the software development phase. For instance, the domain knowledge of database can be more important then the graphic interface in the Analysis phase. So, they will have a value or weight to represent this importance. However, in the Implementation phase, the domain knowledge of graphic interface is more then the database. Then, the rules would cover also the phases of development besides the domain knowledge. After all, the database with the information about the knowledge would contain three different weights to be used depending on the development period. The prioritization could be applied in different ways in each development phase. Moreover, this could be very useful in the planning of the development, because the schedule could be specified with much more accuracy by taking into account this additional information.

Appendix A

In this Appendix all defined use cases to the *Message Exchange System* are given. They are categorized by the module. Each use case is briefly described to give its general idea and purpose. The description gives the main functionalities of each use case.

Use Cases:

Module 1: Processing Data

User Case 1:
Name: Reading Data
Description: Reads charged data per client from the communication medium. The data is received from the client is load into the system. Data validation is executed to check if the message is valid.

User Case 2:
Name: Calculating Cost
Description: Calculates the cost of exchange per client. Every message exchange has a cost. All the time that a message is received and sent the value of the operation is calculated. The costs defined by the system are store for future measurements and statistics.

User Case 3:
Name: Converting Data
Description: Converts the charged data with respect to the client's format. The system can support different kinds of formats to attend different formats from the clients. The data charged is converted to an intermediate format before converting to the format of the destination. The intermediate format is to standardize the process and to make easier the transformation to the format of the client that will receive the message.

User Case 4:
Name: Filtering Data.
Description: Dispatches the converted data to the right client. The destination is identified in the charged data. The message is redirect to the correct receiver.

User Case 5:
Name: Finding Format
Description: Determines the correct format to apply to the data. The format required for the user that will receive the message is detected based on the information about the client.

User Case 6:
Name: Formatting Data
Description: Applies the appropriate transformations to the charged data based on client's format. The formatted used in this transformation is identified by the use case *Finding Format*. If some problem occur formatting data, it is logged.

User Case 7:
Name: Buffering Data
Description: Buffers the formatted data to be transmitted. The message is buffered before sending and the data is kept until the message is completely send. If an overflow error occur, it is logged.

User Case 8:
Name: Wrapping Data
Description: Wraps formatted and buffered data before sending. The data is wrapped in a structure to facilitate the sending.

User Case 9:
Name: Rerouting Data
Description: If the message can not be sent to the correct receiver, it is reroute to the sender.

User Case 10:
Name: Sending Data
Description: Sends the output data to the correct client destination. The output data is the formatted message that was buffered and wrapped.

User Case 11:
Name: Output Data Validation
Description: Checks the output data. Compares the data flow checking incoming and outgoing data. It validates if the message that will be sent was charged into the system and if the receiver is correct. The validation is logged for future measurements, statistics and controls.

Module 2: Logging Data

User Case 12:
Name: Logging information
Description: Logs information during execution. Relevant information to the system is registered for future analysis or measurements.

User Case 13:
Name: Logging Input Data
Description: Logs all the incoming data per client from the communication medium. With the message, the data and time that was received and the client identification is stored. An identification to the log, a number generated by the system, is also stored in the system.

User Case 14:
Name: Archiving Output Data
Description: Archives the output data per client. All the sent messages are archived with the respective information about the client. Date and time that the message is sent are also store.

User Case 15:
Name: Logging Error Messages
Description: Logs the error messages from the process. Every error that occurs during the process is stored. The log contains log identification, the error message, date and time when it occurred, client identification and error type. Log identification is a number generated by the system. Error type indicates the part of the process where the error occurred.

Module 3: Monitoring

User Case 16:
Name: Monitoring
Description: Monitors the information in the system. It defines the information to be monitored.

User Case 17:
Name: Errors Control
Description: Controls the errors that happen during execution. It defines the different kinds of errors handed by the system.

User Case 18:
Name: History Manager
Description: Controls the historic logs. All the logs archived can be accessed and visualized. Stored logs can be searched by the information that they contain. Deletes old archived logs based on defined criterion. The criterion can be a date or a client defined by the system administrator.

User Case 19:
Name: Monitoring Logs
Description: Controls the current logs. Allows the visualization of the registers. Filters logs based on defined criteria. The available criteria in the filters are the information contained in the logs as: log identification, type, message, date, time and client. The details of search result logs can be consulted.

User Case 20:
Name: Monitoring Errors
Description: Controls the current error messages. Allows the visualization of the registers. Filters error messages based on defined criteria. The available criteria in the filters are the information contained in the logged error messages as: error log identification, error type, error message, date, time and client. The details of search result can be consulted.

User Case 21:
Name: Monitors Manager
Description: Graphic interface for controlling all the monitoring. Shows graphically the information controlled by the monitors as the monitor of logs and monitor of errors.

User Case 22:
Name: Error Handling
Description: Controls the errors per client and per process stage. The errors that occur in the process are monitored. If some kind of error occurs a certain number of times, a warning message is sent to the system administrator. Or if a critical error occurs, a message is also sent to the system administrator. The number of messages or the critical messages are defined by the system administrator.

Module 4: Measurements

User Case 23:
Name: Measurements
Description: Measures data and information in the system. It includes the activities to measure the data and information in the process. The measurement can be applied to the logged data or data during the execution.

User Case 24:
Name: Measuring Input Data
Description: Measures the input data per client. Measure the number of messages received per client and per a determined period of time. Measure the size of the received messages.

User Case 25:
Name: Measuring Formatted Data
Description: Measures the formatted data, in the intermediate format. The measurement is done per client and per a determined period of time.

User Case 26:
Name: Measuring Output Data
Description: Measures the output data per client. Measure the number of messages sent per client and per a determined period of time. Measure the size of the sent messages.

User Case 27:
Name: Measuring Interpreted Data
Description: Measures the interpreted data per client, per format and per period of time.

User Case 28:
Name: Measurement Manager
Description: Controls all the measurements and statistics. A graphic interface to visualize and control the measurements and statistics. Allows defining the period or time used in the measurements.

User Case 29:
Name: Measurement Costs
Description: Measures the exchange costs per client and per period of time.

User Case 30:
Name: Statistics
Description: Applies statistics to the data in the system. The statistics can be applied to the logged or measured data. It defined the different kinds of statistics in the system and the possible parameters as determined period of time, client and kind of error.

User Case 31:
Name: Generating Measurement Statistics
Description: Generates measurement statistics based on defined criteria. The criteria are all the available measurements as input data, formatted data, output data and interpreted data. Besides the different kinds of measurements other criteria can be used: per client and period of time. Executes the requests to the generation of the statistics.

User Case 32:
Name: Generating Measurement Reports
Description: Generates reports to be impressed about the measurements based on defined criteria. The criteria are all the available measurements as input data, formatted data, output data and interpreted data. Besides the different kinds of measurements other criteria can be used: per client and period of time. Executes the requests to the generation of the statistics.

User Case 33:
Name: Generating Error Statistics
Description: Generates error statistics based on defined criteria. The criterion is all the available kind of errors. Besides the different kinds of errors other criteria can be used: per client and period of time. Executes the requests to the generation of the error statistics.

Module 5: Protocols and Formats

User Case 34:
Name: Adding Interpreter
Description: Adds a new interpreter to new client's format. New interpreters can be added to the system when this is running, because the system can not stop.

User Case 35:
Name: Defining Protocols
Description: Defines the protocols in the system. The protocols are in the beginning and in the end of the process, where the messages are received and sent. The message received by the system has to be in one of the defined protocols in the system as the message that will be sent.

User Case 36:
Name: Adding Protocol
Description: Adds new protocols in the system. New protocols can be defined to attend the clients.

User Case 37:
Name: Defining Formats
Description: Defines the formats that are used in the transformations. The system can only work with the defined formats.

User Case 38:
Name: Defining Standard Format
Description: Defines the standard format that is used in the transformations. There is a standard format that all the messages charged by the system are converted to this format before converting to the final format. This makes much easier the transformation, because the transformation to format of the user that will receive the message is always done from the standard format. It is not necessary to define transformation form every format to every format.

Appendix B

This Appendix has the aim of explaining terms used in the *Message Exchange System* example. The definition of these concepts helps to understand the use cases.

Concepts:

Charged Data is the data that is load into the system. It is received from the communication medium.

Client is the person that uses the service of exchanging messages. The client can send or receive a message.

Communication medium is the medium used to send or receive messages, this could be a mobile phone for instance.

Converted data is the data in the standard format. The data received from the client, in the client format, is converted to a standard format.

Cost of exchange is the cost of the message transmission. All message exchange has a cost, defined by the company responsible to the transmissions.

Data is all the information running in the system. The data pass through the system in different formats, depending on the moment of the process.

Filters are used to make searches, and criteria are defined in the filters to execute the search. The criteria used in the filters depend on the element that is being searched.

Format is to define how the data is represented. It describes how the data has to look like.

Formatted data is the data in the client's format. The data is represented how defined in the specified format.

Interpreter is to recognize and transform a message in a certain format. It transforms the data in the standard format to the client format.

Log is a stored data with data and time information when this was stored. It is a registry of what happens or runs into the system.

Overflow is when the data flows over the defined limits. This means that the limit to store is exceeded.

Protocol is a standard set of rules that determines how devices communicate with each other. Protocols describe both the format that a message must take and the way in which messages are exchanged.

Appendix C

This appendix contains an overview of the Tool User Interface.

Figure 36 shows the menu bar of the Prioriting Tool. The tool has the main following functionalities:
- Load the data
- Fill the matrix
- Prioritizing
- Show the results
- Show the consistency index
- Show the graph with the results

Each one is detailed in the next sections.

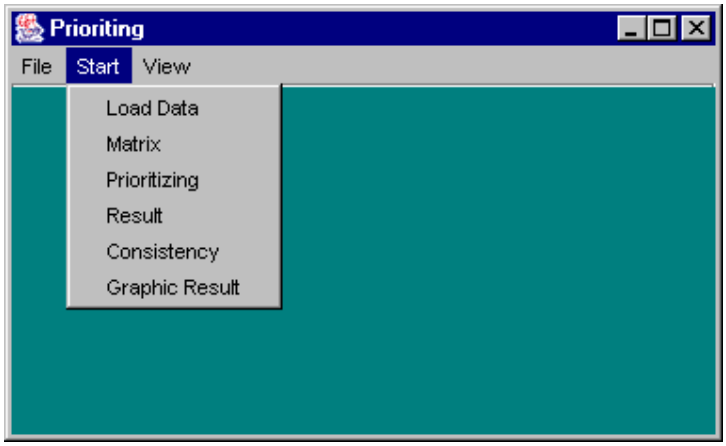


Figure 36. User Interface.

C.1 Load Data

The first action in the system is to load the input data. The other options have no function without loaded data. The input data is a Rational Rose file, this means files with the extension equals .mdl. This file must contain the requirements and/or use cases that need to be prioritized. The requirements and use cases are represented by Use Case entities in Rational Rose. The relationships between the use cases need to be represented in the file. The relationships may be the several possible standard relationships among use cases as generalization, association, extend and include.

Figure 37 illustrates the kinds of relationship among use cases. The use cases in the Order Example are only to demonstrate the user interface. Very few requirements were chosen to facilitate the visualization and comprehension.

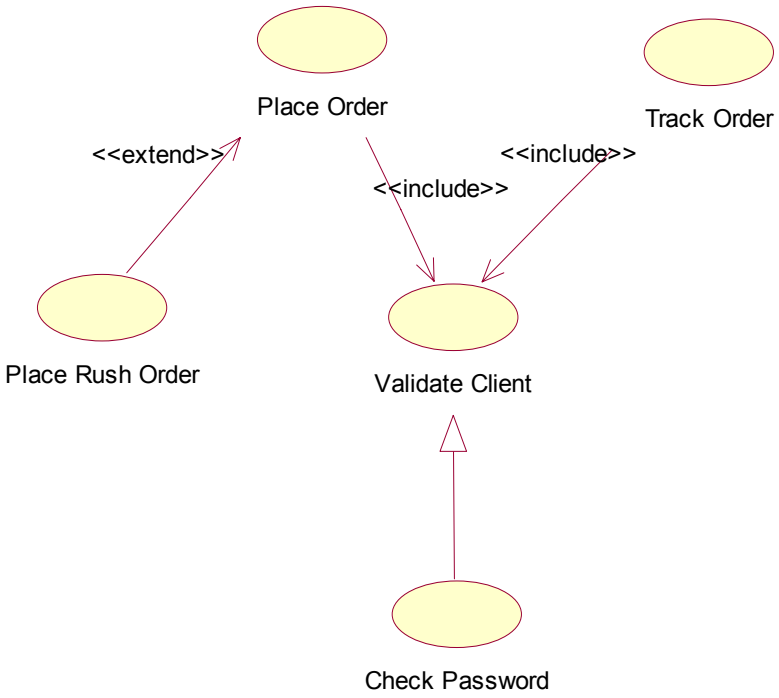


Figure 37. Order Example use case diagram.

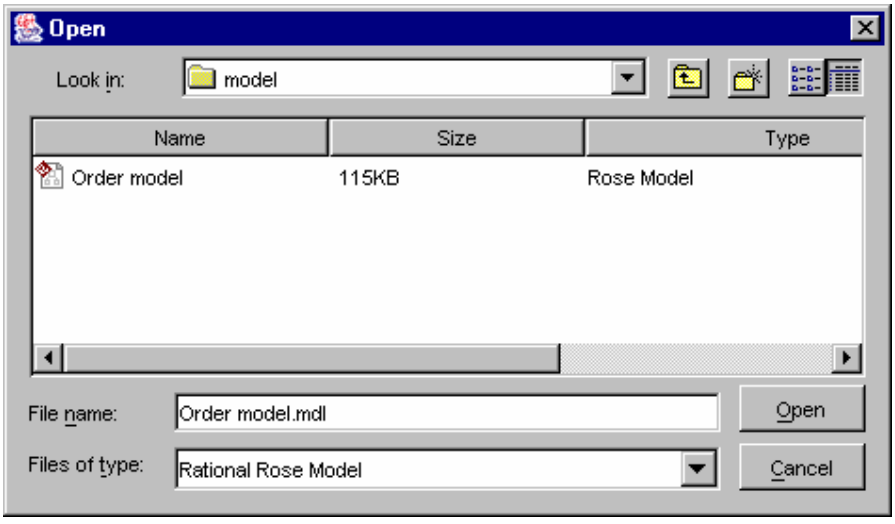


Figure 38. Choosing the Rational Rose Model.

When the chosen file by the user is loaded, the data is parsed and transformed in objects. These objects are called Snapshots and all information is stored in the snapshots. Information means use case name and relationships.

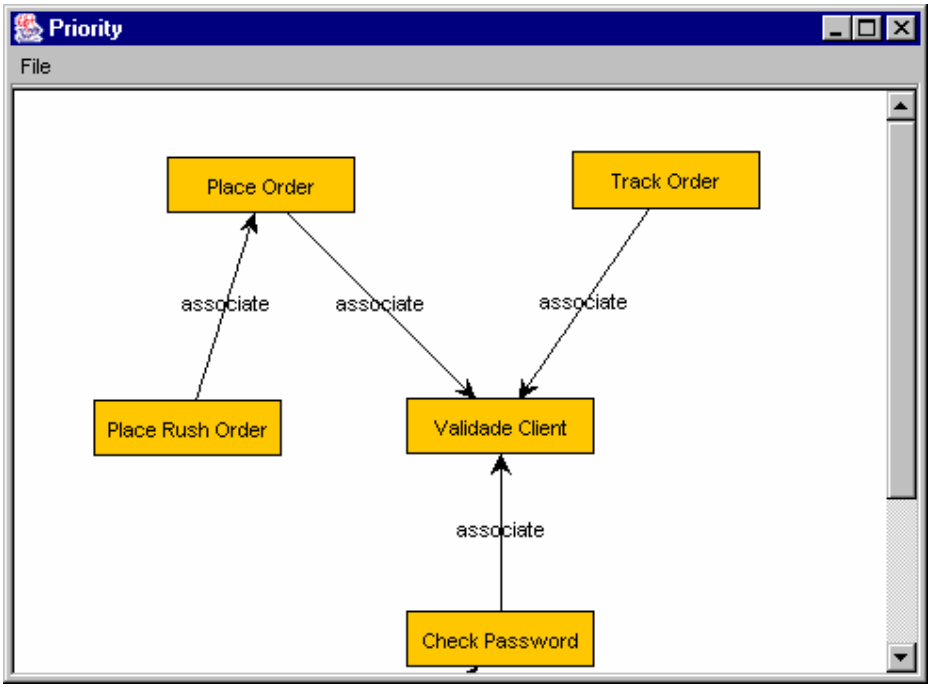


Figure 39. Graph representation of the use cases in the tool.

The requirements are shown in the scream in graph format. Each rectangle represents a requirement or use case and the lines represent the relationships among the use cases. Figure 39 shows an example.

The levels of the requirements to build the hierarchy are defined after loading the use case diagram.
When the graphs are shown, the user has to arrange the requirements in the hierarchy. The user can drag and drop the requirements to the desired position to define the levels.

Looking to Figure 37 and 39, it is possible to verify the result of the transformation.
The levels are defined by the system depending on the position of the elements in the scream. Each connection present in a use case is analyzed. The element that is in a lower position is dependent on the other.
Figure 40 illustrates how to see the levels. The use case *Validate Client* is in a lower position then the use case *Track Order*. So, *Validate Client* is dependent on *Track Order*.
As *Track Order* is in the top level, it belongs to the level number 1. So, as *Validate Client* is under *Track Order*, it belongs to the level number 2.

If two elements that have some relations between them are located exactly in the same high or in the same location vertically, they are situated in the same level.

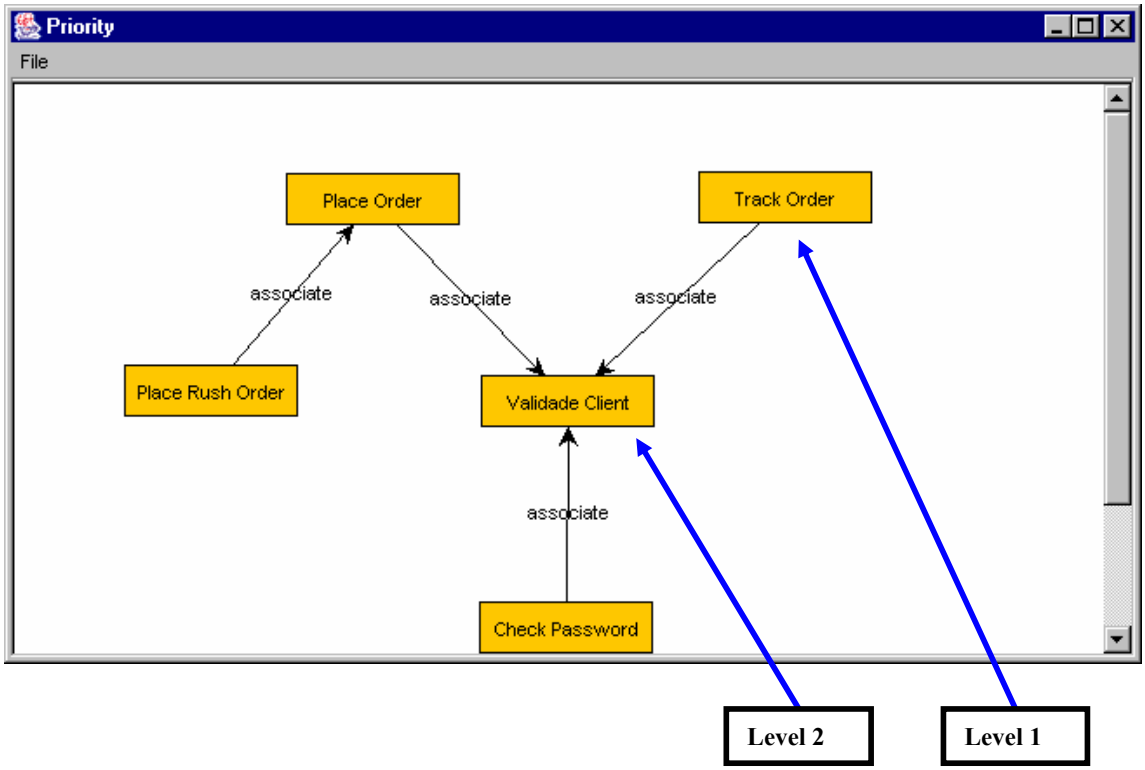


Figure 40. Defining the hierarchy.

C.2 Matrix

After loading the data, the second step is to fill the matrix. The system builds a matrix with the elements in the top level. All the use cases in this level should be judged. The name of the use cases is presented in the first column and first row in the matrix. Only the upper part of the matrix, above the main diagonal, needs to be filled. The main diagonal is already filled by the system with value 1. The lower part of the matrix, under the main diagonal, doesn't require values filled by the user. These values are calculated by the system with the reciprocal values.
Figure 41 has an example of hierarchy to illustrate the matrix that is build. Figure 42 shows the matrix, as there are three requirements in the top level, these three requirements are in the matrix.
Figure 43 shows the matrix with the values of the relative comparison. These values are defined just to illustrate the example.
All the process about how to fill the matrix is explained in section 2.4.3.

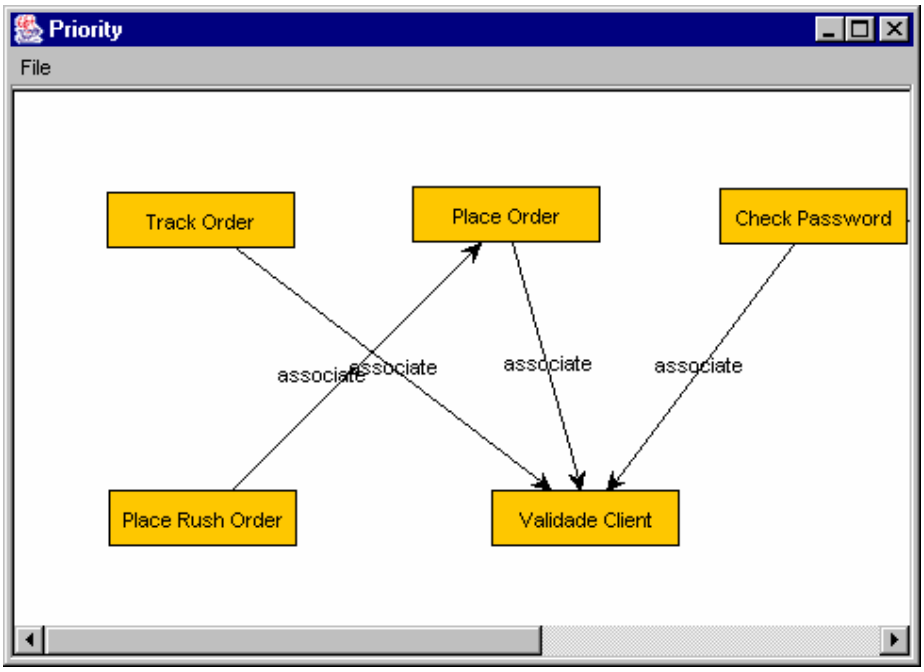


Figure 41. Hierarchy example.

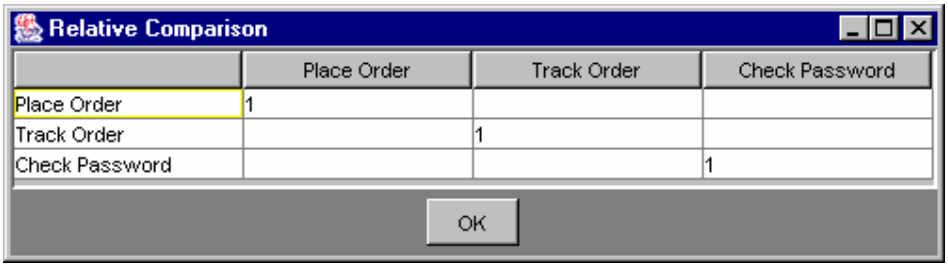


Figure 42. Matrix.

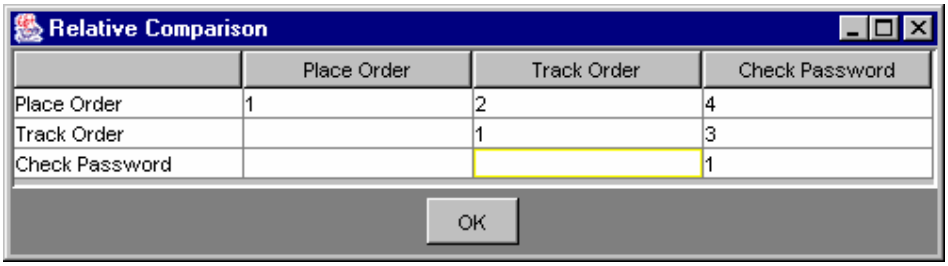


Figure 43. Filled Matrix.

C.3 Prioritization

This is the step where the process of prioritization is started up and executed.
First, the information acquired when the Rational Rose file is loaded that is represented in the snapshots is transformed to the Items.
The values of the relative comparisons filled by the user in the matrix are also transformed into Comparisons.
The classes Items and Comparisons were explained in section 5.3.1.
With all the necessary information inside the system, the process is started.
There are several steps that are executed to all the levels. The only difference among the levels is between the first level and the other. The values that are used to calculate the priority to the first level are extracted from the comparisons and to the other levels are extracted from the dependencies.
One level is processed for each time.
Primarily the matrix is formed with the values. Then the reciprocal values are calculated.
After that the verification to check if there are some positions in the matrix that were not informed. This process described in section 4.4 is applied to solve the null values.
If the minimal number of values is not informed a message is showed to warn the user.
Finally the matrix is ready for the calculation.
The calculations described in section 2.4.3.2 are executed. The result is stored.
The consistency index is calculated as defined in section 2.4.3.3.

C.4 Results

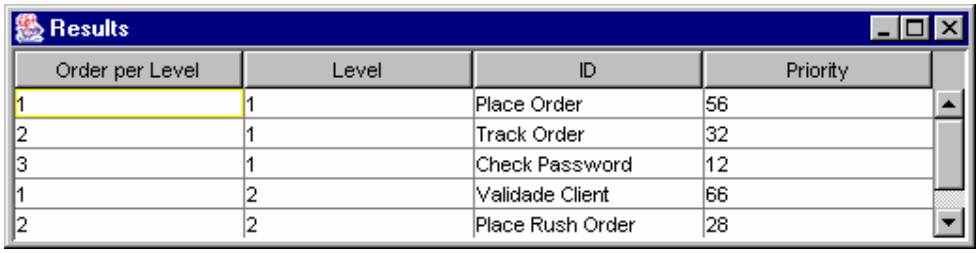


Figure 44. Table of results.

This option allows the user to see the results obtained in the calculation.
A table is shown with all the requirements involved in the process.
The first columns has the order per level, this means the order that the element is in its level.
The second column indicates the level that the element is.
The third column brings the name of the requirement, then they can be recognized.
The fourth column presents the priority values.
These values are expressed in percentage.
The sum of each level is 100%.

C.5 Consistency

The option of consistency brings the consistency indices calculated in the process.
A table is shown with all the levels.
The consistency index is calculated for each matrix, this means for each level there is a consistency index.
In the table, the first column has the level of the index. The second column has the value of the inconsistency.
It is important to evaluate the value of the first level. As describe in section x, if the consistency index is larger than 0.1, this means that the result is not so good.
So the relative comparisons should be reviewed.
If this happens, a message is shown to the user to alert that the matrix should be analyzed again.
Analyzing the values insert in the matrix, these values can be changed as the user wishes.
After that the prioritization should be executed again. New results are provided including new consistency values.
The user has to check the new values and verify if now the consistency is coherent.
If the value obtained now is less then 0.1, the result is consistent. But if not, the process can be executed again. The advice described in paragraph 2.4.3.3.4 can help in this task to find the inconsistencies.

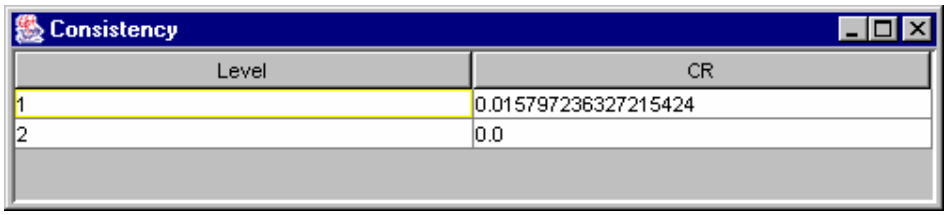


Figure 45. Table of consistencies.

C.6 Graphic Results

Graphic results option shows the graphs with the requirements, in the same position chosen by the user to define the hierarchy.
It is possible to check the priority value obtained in the calculation.
Positioning the mouse in the desired requirement, the value is presented in a rectangle, called tool tip. Figure 46 shows this particularity, the use case *Track Order* has the value *32* as the priority.

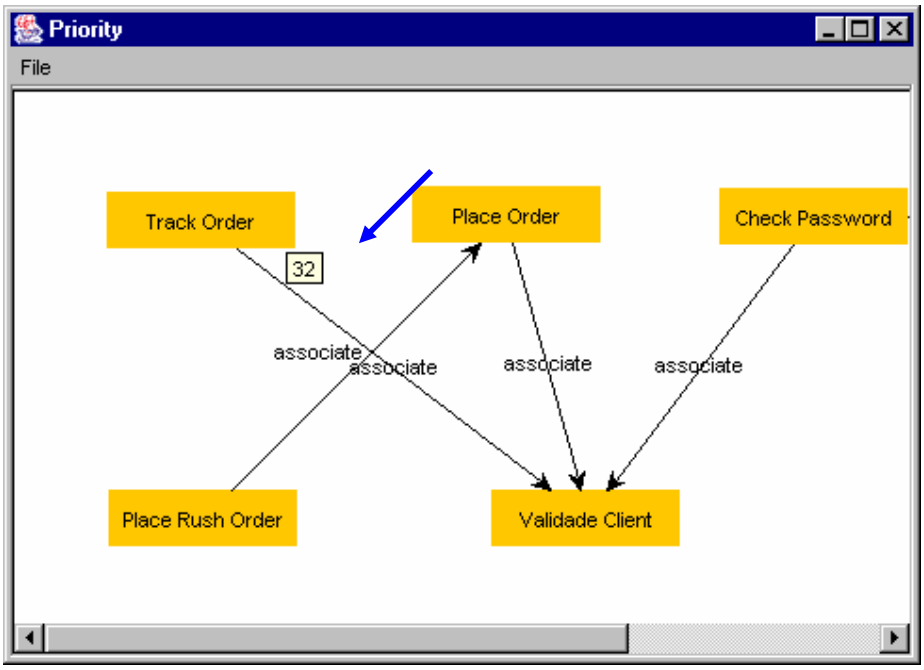


Figure 46. Graphic Result.

References

[CHI03] Chitnis, M., Ananthamurthy, L. and Tiwari P., UML Part 7: Activity Diagram, 2003

[DAV93] Davis, A., *Software Requirements: Objects, Functions and States*, Prentice-Hall 1993.

[ECL] Eclipse, http://www.eclipse.org.

[GAM95] Gamma, E., Helm, R., Johnson, R. and Vlissides, J., *Design Patterns, Elements of Reusable Object-Oriented Software*, Addison-Wesley Publishing Company, Inc., Massachusetts, 1995.

[GLA03] Glandrup, M. and Mehmet, A., *Using Problem Patterns for Mapping Requirements to Technical Problems*, Enschede, 2003.

[GOL89] Golden, B.L., E.A. Wasil and P.T. Harker (eds.) (1989), *The analytic hierarchy process, applications and studies*. Berlin, Springer-Verlag.

[GUS02] Gustavsson, R., Ala-Kurikka, J. and Rulli, S., *Domain Specific Design Patterns - A report in the course Object-Oriented Programming Advanced Course*, Mälardalen University 2002.

[HAE83] Haerder, T. and Reuter, A., *Principles of Transaction-Oriented Database Recovery*, ACM Computing Surveys, Vol. 15, 1983.

[HAR87a] Harker, *P., Incomplete Pairwise Comparisons in the Analytic Hierarchy Process*, Mathematical Modelling, 1887.

[HAR87b] Harker, P., *Alternative Modes of Questioning in the Analytic Hierarchy Process*, Mathematical Modelling, 1887.

[IEE90] IEEE Std 610.12-1990. *IEEE Standard Glossary of Software Engineering Terminology*, The Institute of Electrical and Electronics Engineers, New York, 1990.

[JAC92] Jacobson, I., Christerson, M., Jonsson, P., vergaard, G., Object-Oriented Software Engineering: A Use Case Driven Approach, Addison-Wesley, Harlow, England, 1992.

[JAC99] Jacobson, I., Booch, G and Rumbaugh, J., *The Unified Software Development Process*, Addison-Wesley, 1999.

[JAVa] Java, http://java.sun.com.

[JAVb] Java 2 Platform, Standard Edition, v 1.4.1 API Specification, http://java.sun.com/j2se/1.4.1/docs/api/.

[JGR] JGraph, http://www.jgraph.com.

[KAR97] Karlsson, J. and Ryan, K., *A cost-value approach for prioritizing requirements*, IEEE Software, pp. 67-74, Sep-Oct 1997.

[MIL01] Miller, G., *Java Modeling: A UML workbook, Part 3, User interface logic in use case modeling*, 2001.

[NOP02] Noppen, J., Tekinerdogan, B., Aksit, M., Glandrup, M. and Nicola, V., *Optimising software development policies for evolutionary system Requirements*, 2002.

[NOP03] Noppen, J., Aksit, M., Nicola, V. and Tekinerdogan, B., *Scheduling software development process for evolving market demands*, Enschede 2003.

[OMG01] OMG Unified Modeling Language Specification (version 1.4), 2001. Available from www.omg.org.

[PAR99] Park, J., Port, D. and Boehm, B., *Supporting Distributed Collaborative Prioritization for WinWin Requirements Capture and Negotiations*, Center for Software Engineering, Los Angeles 1999.

[POR98] Port, D., *Derivation of Domain Specific Design Patterns. USC Center for software engineering*, 1998.

[PUT94] Puterman, M., *Markov Decision Process, Discrete Stochastic Dynamic Programming*, 1994, Wiley-Interscience.

[PRE97] Pressman, R., *Software Engineering, A Practitioner's Approach*, McGraw-Hill, 1997.

[RAT] Rational Rose, http://www.rational.com.

[RUM98] Rumbaugh, J., Jacobson, I. and G. Booch, *The Unified Modeling Language Reference Manual,* Addison-Wesley Publishing Company, 1998.

[RYA97] Ryan, K., Karlsson, J., *Prioritizing Software requirements in an Industrial Setting*, in International Conference on Software Engineering (ICSE'97).

[SAA93] Saaty, J., *Expert Choice User Manual*, Expert Choice, Inc., McMean, VA, 1993.

[SAA72] Saaty, T. L., An Eigenvalue Allocation Model for Prioritizing and Planning, Energy Management and Policy Center, University of Pennsylvania, 1972.

[SAA77a] Saaty, T. L., The Sudan Transport Study, Interfaces, vol. 8, 1977.

[SAA77b] Saaty, T. L. and Bennett, J. P., Fancing Tomorrow's Terrorist Incident Today, U.S. Department of Justice, LEAA, 1977.

[SAA79] Saaty, T.L., Mariano, R., *Rationing Energy to Industries: Priorities and Input-Output Dependence*, Energy Systems and Policy 3, 1979.

[SAA80] Saaty, T.L., *The analytic hierarchy process*, McGraw-Hill, New York; 1980.

[SHA95] Shaw, M. and Garlan, M., *Formulations and Formalisms in Software Architecture, Volume 100 – Lecture Notes in Computer Science*, Springer-Verlag, 1995.

[TEK00a] Tekinerdogan, B. and Aksit M., *Syntesys-Based Software Architecture Design,* Print Partners Ipskamp, Eschede 2000.

[TEK00b] Tekinerdogan, B. and Aksit, M., *Separation and composition of concerns through synthesis-based design*, Advanced Separation of Concerns (ASC'00) workshop at the ACM Conf. on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA 2000), October 2000.

[TRE02] TRESE, University of Twente, *Systensis-Based Software Architecture Design Homepage*, http://trese.cs.utwente.nl/architecture_design/

[WIE96] Wiegers, K., *Creating a Software Engineering Culture*, Dorset House Publishing, New York, 1996.

[WIE99] Wiegers, K., *First Things First: Prioritizing Requirements*, Software Development, September 1999.

[ZON03] Zonneveld, H., *A software architecture design for the Portable Storage Container – The design of a software architecture with SYNBAD*, Eindhoven 2003.