Software Engineering Research Group
Final thesis

Philips Medical Archive Splitting

Marco Glorie

Software Engineering Research Group
Final thesis

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

SOFTWARE ENGINEERING

by

Marco Glorie
born in Heemskerk, the Netherlands

Software Engineering Research Group
Department of Software Technology
Faculty EEMCS, Delft University of Technology
Delft, the Netherlands
www.ewi.tudelft.nl

MR Department
Philips Medical Systems
Veenpluis 6
5684 PC Best, the Netherlands
www.medical.philips.com

Software Engineering Research Group
Final thesis

Author: Marco Glorie
Student id: 1133772
Email: marcoglorie@hi.nl

Abstract

Philips Medical Systems makes MR-scanners - among many other products - for medical purposes such as diagnostics. MR stands for Magnetic Resonance and the technology enables MR-scanners to analyze the human body for potential diseases. The software for these MR-scanners is complex and has been evolving for more than 27 years. The software archive has been evolved into a single 9 MLOC archive, which is now hard to maintain. This paper aims at ways to split the single archive into multiple archives that can be developed independently. We will discuss formal concept analysis and cluster analysis as analysis methods to obtain such archives from the single archive, using the Symphony framework. We will conclude with an evaluation of the used analysis methods.

Thesis Committee:

Chair:	Prof. Dr. A. van Deursen, Faculty EEMCS, TU Delft
University supervisor:	Dr. A. Zaidman, Faculty EEMCS, TU Delft
Company supervisor:	Drs. L. Hofland, Philips Medical Systems

Preface

This document would not have been possible to create without the help from both the side of Philips Medical Systems and Delft University of Technology. Therefore I would like to thank Lennart Hofland for his support, advices and reviewing activities. Also I thank Andy Zaidman and Arie van Deursen for supporting me with reviews and advices on the subject.

Further during my project I was surrounded by colleagues from the ‘image processing and platform’ group. I really appreciated they invited me to their group meetings, this gave me a good insight of how they work at Philips and what difficulties they are facing. Also I enjoyed the several group trips, such as our ‘adventurous’ GPS-tour in Velthoven. Thanks for that.

Finally my thanks also go to Mulo Emmanuel, Christian Schachten, Jihad Lathal and Frederic Buehler, who all made my stay in Best an exciting stay.

Marco Glorie
Best, the Netherlands
August 3, 2007

Contents

List of Figures

Chapter 1

Introduction

Philips Medical Systems (PMS) develops and produces complex systems to aid the medical world with monitoring, diagnostic and other activities. Among these systems is the MR-scanner, a product that uses Magnetic Resonance technology to analyze the human body for potential diseases. The software for these MR-scanners is complex and has been evolving for more than 27 years. The system is a combination of hardware and software and contains (real-time) embedded software modules. Many software technologies are used and also third party off-the-shelf modules are integrated in the software.

Currently the software is developed using branching with a single 9 MLOC software archive consisting of approximately 30,000 files. Merging the multiple software streams causes integration problems and the many dependencies among the files make that the feature that takes the longest time to be developed determines the release time of the complete project.

This way of developing is the result of many years of software evolution, but the MR department of PMS sees the current process of development as inefficient and wants to improve the efficiency of the process. A team has been formed to improve the development process by evolving the current architecture in a new architecture that can be more easily maintained.

The team sees the new architecture as an architecture consisting of about 7 archives that can be developed independently. In order to obtain these independent archives the current software archive should be analyzed on what modules should form these archives and clear interfaces should be defined between the archives.

We use Symphony as an architecture reconstruction framework to obtain a splitting of the current archive, using analysis methods available in literature. Existing algorithms and tools are discussed and an approach is presented to use the analysis methods to obtain a splitting of the current archive.

The following outline is followed in this thesis: in chapter 2 we present the MR case and research questions which are extracted from the MR case. The research questions will be answered in the following chapters. In chapter 3 we discuss Symphony, which is used as framework for the analyses. Then in chapter 4 we discuss formal concept analysis and in chapter 5 cluster analysis, as analysis methods to obtain a splitting of the archive. Finally in chapter 6 conclusions will be drawn from the researched matters.

Chapter 2

Problem definition

In order to define the problem definition of this thesis first the MR case is discussed. From the MR case the problem definition is extracted. In section 2.1 the MR case is discussed and in section 2.2 the problem definition of this thesis is presented.

2.1 The MR case

Philips Medical Systems (PMS) develops evolutionary medical systems such as MR-scanners, based on Magnetic Resonance technology. The medical activities of Philips date back to 1918, when it introduced a medical X-ray tube. Today PMS is a global leader in diagnostic imaging systems, healthcare information technology solutions, patient monitoring and cardiac devices.

The MR case concerns the software for the MR-scanners produced by PMS. The software is developed on multiple sites (Netherlands, USA and India) and there are about 150 developers working on the software spread out over these 3 sites. Currently, to minimize project risks, parallel software streams are used (branching).

Current development process The current way of developing the software for MR-scanners is seen by the MR department as inefficient. A first reason for this inefficiency is that because multiple software streams are used, at some point in the development process, these need to be merged.

To give a basic overview of how this works: The streams work with copies of that parts of the software archive that are to be modified by developers. When two streams are merged it is possible that there is an overlap in changed sections of the archive copies. This process of development gives a lot of merging problems.

A second reason is the fact that there are many dependencies of the building blocks in the archive, which cause that the release moment is determined by the release moment of the slowest feature.

The ‘Software Architectuur Team’ (SWAT) of MR is responsible for providing a strategy to cope with the current problems. The team has created a vision that the future software archive should consist of about 7 independent archives. Each of these different archives

should act as a component and they should be combinable with each other independently. In order to implement this vision the current archive should be split into pieces.

System's characteristics and building block structure To provide some insight into the MR case, an overview of some of the system's characteristics is presented. The software for MR-scanners has been evolving for more than 27 years. The application-suite is built using a lot of different software technologies, such as C, C++, C# and Perl and many parts of it can be considered as legacy software. Also, many off-the shelf components are integrated into the system. This has resulted in the current situation, wherein the code base consists of approximately 9 MLOC, located in around 30,000 files.

Files are contained in entities called *building blocks*; the files are subdivided into nearly 600 building blocks. Furthermore, a lot of dependencies exist between these building blocks.

The code archive is organized by structuring building blocks in a tree-structure, a so-called *building block structure* or *building block hierarchy*. At the highest level of abstraction in this building block structure there are the so-called *subsystems*, which contain multiple lower level building blocks. However, the tree-structure of building blocks does not represent the high level architecture of the system. Figure 2.1 shows the hierarchy of building blocks.

There exists detailed documentation on the architecture, such as UML diagrams and project documentation. More details on the architecture can be found in appendix A. For this thesis mainly *project documentation* is used: The documentation specifies the purpose of the projects and - among other details - which building blocks are expected to be under the scope of projects.

For the last two years these scopes have been documented and around 50 documents are available. An example of a description of a scope of a (fictive) project can be seen in figure 2.2.

2.2 Problem definition

In order to achieve a splitting of the archive for the MR case, the software should be analyzed in order to obtain archives that can be developed independently. Therefore the following research question is central in this thesis:

- Which analysis methods are available in order to split an archive into multiple independent archives?

To come to a solution for this problem, we envision that we should first refine our central research question in the following subsidiary research questions:

- Are there tools or algorithms available to perform the 'splitting' analysis?
- How can the analysis methods be applied to the MR case?

Building Block (BB) structure				
Subsystem_1				
	BB_1_1			
		BB_1_1_1		
			BB_1_1_1_1	
				File_1
				...
				File_n
			BB_1_1_1_2	
				...
		BB_1_1_2		
		...		
	BB_1_2			
	...			
Subsystem_2				
...				
Subsystem_n				

Figure 2.1: The building block structure

Scope project: 'new_image_printer'		
Affected building blocks	*Why and what*	*Remarks*
ExportImage	Include new type printer for export	-
PrinterSelect	Add new printer type for selection	-
PrinterConfiguration	Add parameters for new printer	-

Figure 2.2: The scope of a fictive project

- Are these analysis methods scalable enough to cope with large scale industrial software?

We discuss *formal concept analysis* and *cluster analysis* as analysis methods in order to split the software archive in multiple independent archives. To apply the analysis methods algorithms or tools performing the analysis are required. Therefore existing algorithms and tools are discussed for this purpose.

To apply the analysis methods to the MR case, first we discuss how the analysis methods are applied in the available literature and an approach is presented how to come to a splitting of the current archive of MR.

As the archive for the MR case concerns a large scale industrial software archive, scalability issues have to be taken into account. Therefore scalability issues of applying analysis methods for the MR case are discussed and approaches are proposed to cope with these issues.

After performing analyses to the MR case with the researched analysis methods, results are evaluated with the domain experts at PMS to validate the outcomes. Finally a conclusion will be drawn from the researched matters.

Chapter 3

Symphony

The problem of how to obtain multiple independent archives from a single archive can be seen as an architecture reconstruction problem. *Symphony* is a view-driven software architecture reconstruction method and is used in this paper to provide a framework to come to a solution of the defined problem. [32]

Central to Symphony are views and viewpoints to reconstruct software architectures. The terminology of views and viewpoints are adapted from the IEEE1471 standard. The *view* is a "representation of a whole system from the perspective of a related set of concerns". A view conforms to a *viewpoint*. A viewpoint is a "specification of the conventions for constructing and using a view." [25]

Symphony is a framework describing the process of architecture reconstruction. The Symphony process consists of two stages: [32]

1. Reconstruction Design

2. Reconstruction Execution

The two stages are typically iterated because reconstruction execution reveals information about the system that can serve as an input for a better understanding of the problem and a renewed reconstruction design. In section 3.1 the reconstruction design stage is discussed and in section 3.2 the reconstruction execution. Finally in section 3.3 is discussed what Symphony means for the MR case.

3.1 Reconstruction design

The first stage of Symphony is the reconstruction design. During this stage the problem is analyzed, viewpoints for the 'target views' are selected, 'source views' are defined and 'mapping rules' from source to target views are designed. [32] These terms are explained below.

The *source view* is a view of a system that "can be extracted from artifacts of that system, such as source code, build files, configuration information, documentation or traces". [32]

The *target view* is a view of the system that describes the “as-implemented architecture and contains the information needed to solve the problem/perform the tasks for which the reconstruction process was carried out”. [32]

The *mapping rules* define how the target view is to be created from the source view and depends on for example what analysis method is used. When during the reconstruction design the problem, views and mapping rules are defined the next step is to perform the reconstruction execution.

The reconstruction design stage consists of the *problem elicitation* and the *concept determination*. During the problem elicitation the problem that needs to be solved is analyzed.

Having analyzed the problem, the concept determination follows, where concepts relevant to the problem are identified and a recovery strategy is presented: the target viewpoint, source viewpoint and mapping rules are defined or refined.

So during the first stage of Symphony - the reconstruction design - the problem is analyzed and the viewpoints and mapping rules are defined or refined. Refined, because there could be new insights into the problem after the reconstruction execution stage.

3.2 Reconstruction execution

The second stage of Symphony is the reconstruction execution. During the reconstruction execution stage the system is analyzed, source views are extracted and the mapping rules are applied to obtain the target views, earlier defined during the reconstruction design. [32]

This thesis follows the *extract-abstract-present* approach used by Symphony. Symphony presents three steps to apply this approach:

1. Data gathering
2. Knowledge inference
3. Information interpretation

The data gathering step extracts information from the system’s artifacts that can be used to recover selected architectural concepts from the system. These artifacts include the source code, but also build-files or documentation of the system.

During the knowledge inference step the mapping rules, defined during reconstruction design, are applied to the source view to create a target view.

Finally during the information interpretation step the created target views are analyzed and interpreted to solve the problem. Results from this stage lead to a refined problem definition or to a refined reconstruction design to eventually come to a satisfying solution of the problem.

Figure 3.1 - adopted from [32] - describes the three steps of the reconstruction execution. The figure also shows that the knowledge inference step can be repeated, by inferencing knowledge from intermediate results of a previous step of inference.

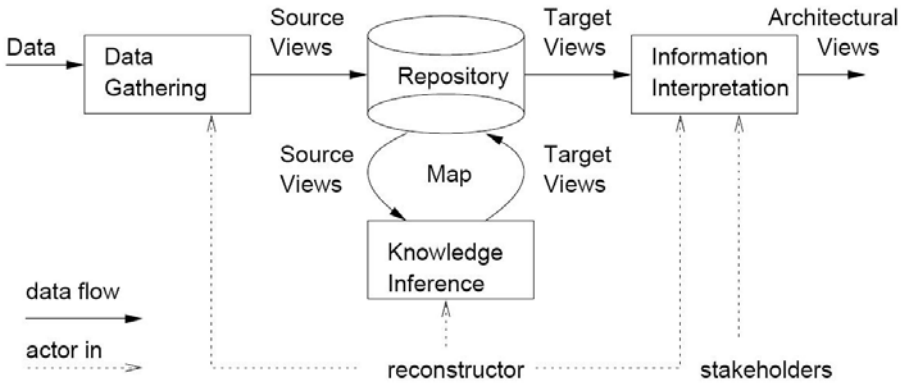

Figure 3.1: Reconstruction execution interactions

3.3 Symphony for the MR case

In this section we discuss how Symphony is used as a framework for the MR case and how the structure of this thesis follows the steps in the Symphony reconstruction process. In section 3.3.1 we discuss the first phase of Symphony: the reconstruction design, applied to the MR case and in section 3.3.2 the second phase: the reconstruction execution.

3.3.1 Reconstruction design for the MR case

The framework of Symphony is used for the MR case because the problem of splitting the single archive into multiple independent archives can be seen as an architecture reconstruction effort. The first stage of Symphony, the reconstruction design, starts with the problem elicitation. This has been defined in chapter 2, the problem definition.

The concept determination of Symphony, as part of the reconstruction design, for the MR case means defining the source views, target views and the mapping rules.

Two different analysis methods are used to analyze how a splitting can be achieved for the MR case: *Formal concept analysis* and *cluster analysis*. For each of the two different analysis methods a different source viewpoint and different mapping rules are defined, which serve as different bases for the reconstruction design. Chapters 4 and 5 describe the two approaches.

Source views For both analysis methods static relations between entities are taken as source viewpoint. Also the existing building block hierarchy is taken as source view for both analysis methods.

Mapping rules The mapping rules differ for both analysis methods. The mapping rules are inherent to the theory of each of the two analysis methods. For formal concept analysis this can be found in chapter 4 and for cluster analysis this can be found in chapter 5.

Target views The target view is identical for both approaches and can readily be selected based on the description of the MR case and the problem definition. The problem is how to split the single archive into multiple independent archives which leads to a *Module viewpoint*. [16] The viewpoint identifies high-level architecture abstractions in the system and describes relationships among them.

3.3.2 Reconstruction execution for the MR case

The processes of analysis proposed in the sections 'Application to the MR case' for both analysis methods follow the extract-abstract-present approach of the reconstruction execution stage of Symphony. For formal concept analysis this can be found in section 4.6 and for cluster analysis this can be found in section 5.5.

Data gathering During the data gathering step static relations between entities are acquired for both analysis methods. In this section is described how these relations are obtained.

A commercial tool called 'Sotograph' is available at PMS to statically extract relations from the code archive. These relations include the following types: aggregation, catch, componentcall, componentimplementation, componentinterfacecall, externalias, externdeclaration, frienddeclaration, inheritance, metadata, throw, throws, typeaccess and write. [31]

The relations are analyzed on the method / function level. Relations on higher abstraction levels - such as the file or building block level - are obtained by accumulating the relations to and from the lower level abstraction levels.

Sotograph only analyses the parts of the archive which are written in C, C++ and the proprietary language GOAL-C. As Sotograph does not provide an integrated view with the C# parts of the archive yet, this thesis is focussed on the parts that are not written in C#. This means that the scope of the analyses in this thesis is limited to around 15.000 files and 360 building blocks, consisting of around 4 million LOC.

Sotograph maintains a database containing information of the static code analysis, including building block hierarchy information. This database can be seen as a populated repository containing the extracted source view.

Knowledge inference and information interpretation For each of the two analysis methods the knowledge inference and information interpretation differ, because of the two different underlying theories. The details can be found in chapter 4 for formal concept analysis and in chapter 5 for cluster analysis.

Chapter 4

Formal Concept Analysis

In order to split the software archive for the MR case the archive should be analyzed to obtain archives that can be developed independently. This section discusses Formal Concept Analysis (FCA) as an analysis method for this purpose.

FCA has been introduced by Wille around 1980. [37] The analysis method is used for structuring data into units. These units are "formal abstractions of concepts of human thought allowing meaningful and comprehensible interpretation". [10]

They state that a concept can be philosophically understood as "the basic units of thought formed in dynamic processes within social and cultural environments". The philosophical view is that a concept is defined by its extension, that is all the objects that belong to the concept and by its intention, that is all the attributes that apply to all the objects of the concept. To be less formal: FCA provides a way to identify sensible groupings of objects that have common attributes. [10]

The following subsections will describe how FCA works. In section 4.1 the term formal context is defined and in section 4.2 the term formal concept. The subconcept-superconcept-relation and the concept lattice is explained in section 4.3 and algorithms to construct the concept lattice from a context are discussed in section 4.4. Section 4.5 discusses applications of FCA and in section 4.6 the applicability to the MR case is explored. Then in section 4.7 the results of the analyses are discussed and the results are evaluated in section 4.8. Finally in section 4.9 a summary is given.

4.1 Formal context

The *formal context* defines what objects have what attributes and is the environment in which formal concepts can be defined. (One should see the terms objects and attributes in an abstract way, in FCA these objects and attributes have no such meaning as in OO-programming for example).

The original mathematical definition is defined using capitals that designate original German words. [10] The definition in this thesis uses capital letters designating the English translations and is as follows:

A triple (O,A,R) is called a *formal context* when O and A are finite sets (the objects and the attributes respectively) and R is a binary relation between O and A that is: $R \subseteq O \times A$. $(o,a) \in I$ is read: object o has attribute a.

In figure 4.1 an example of a context is shown by indicating relations between objects and attributes. This table illustrates a 'crude characterization of mammals' adopted from Siff and Reps. [28]

		attributes				
		four-legged	hair-covered	intelligent	marine	thumbed
objects	cats	v	v			
	dogs	v	v			
	dolphins			v	v	
	gibbons		v	v		v
	humans			v		v
	whales			v	v	

Figure 4.1: Crude characterization of mammals

4.2 Formal concept, extent and intent

A *formal concept* or just called concept is a maximum set of objects that all share the same attributes. The mathematical definition is as follows: [10]

When for $X \subseteq O$ is defined: $X' := \{a \in A | (o,a) \in R \ for\ all\ o \in X\}$ and for $Y \subseteq A$ is defined: $Y' := \{o \in O | (o,a) \in R\ for\ all\ a \in Y\}$ then a pair (X,Y) is a *formal concept* of (O,A,R) if and only if $X \subseteq O$, $Y \subseteq A$, $X' = Y$ *and* $X = Y'$.

In the figure 4.1 can be seen that both the cats as the dogs are four-legged and hair-covered. When we define set X as {cats, dogs} and set Y as {four-legged, hair-covered} then ({cats, dogs}, {four-legged, hair-covered}) is a formal concept of the context of the mammal example. Set X is called the *extent* and set Y the *intent*. [28]

Figure 4.2 shows the extent and the intent of the concepts for the example of the crude characterization of mammals, adopted from Siff and Reps. [28]

4.3 Subconcept-superconcept-relation and concept lattice

Concepts are related to each other when the *subconcept-superconcept-relation* holds, in fact the concepts are naturally ordered by the relation: [10]

$$(X_1,Y_1) \leq (X_2,Y_2) :\Longleftrightarrow X_1 \subseteq X_2 (\Longleftrightarrow Y_2 \subseteq Y_1)$$

top	({cats, gibbons, dogs, dolphins, humans, whales}, Ø)
c5	({gibbons, dolphins, humans, whales}, {intelligent})
c4	({cats, gibbons, dogs}, {hair-covered})
c3	({gibbons, humans}, {intelligent, thumbed})
c2	({dolphins, whales}, {intelligent, marine})
c1	({gibbons}, {hair-covered, intelligent, thumbed})
c0	({cats, dogs}, {hair-covered, four-legged})
bottom	(Ø, {four-legged, hair-covered, intelligent, marine, thumbed})

Figure 4.2: Extent and intent of the concepts for the mammal example

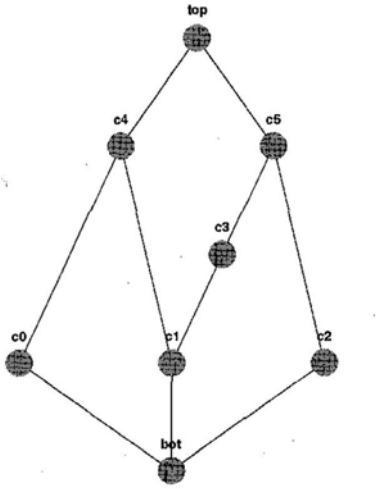

Figure 4.3: Concept lattice for the mammal example

For instance the concept ({cats, dogs}, {hair-covered, four-legged}) is a subconcept of the concept ({cats, gibbons, dogs}, {hair-covered}). The subconcept-superconcept-relation forms a complete partial order over the set of concepts. This ordered set is called the *concept lattice* of the context (O,A,R). This lattice has on top the concept ({all objects}, {no attributes}) and as bottom the concept ({no objects}, {all attributes}). In figure 4.3 the concept lattice of the mammal example is shown, adopted from Siff and Reps. [28]

4.4 Concept lattice construction

To construct a concept lattice for a context several algorithms exist. There are two types of algorithms for computing the concept lattice for a context: batch algorithms and iterative algorithms. Batch algorithms construct the concept lattice bottom-up or top-down. An example of a top-down approach is Bordat's algorithm which starts with the node ({all

objects}, {no attributes}). From this node it generates all the children which are then linked to their parent. This is iteratively repeated for every newly generated pair. [3]

An example of a batch algorithm that constructs the concept lattice bottom-up is Snelting's algorithm. Snelting shows that the worst case running time of the construction algorithm is exponential in the number of elements in the lattice. However Snelting empirically determines that in practice, the concept lattice can typically be constructed in $O(n^3)$ time when using sets of size n for the objects and attributes. [29]

Iterative algorithms do not build the concept lattice from scratch but update the concept lattice incrementally. When adding a new object to the concept lattice depending on the element and its relations some nodes in the lattice will be modified, new pairs are created. These algorithms have been created because the idea was that it will takes less time to update a concept lattice then to build it from scratch. An example of an iterative algorithm is Godin's algorithm. [15]

Godin compares some batch and iterative algorithms to see in what cases the algorithms should be used. Surprisingly he shows his most simple proposed iterative algorithm outperforms the batch algorithms when the number of instances grows. Further he shows empirically that on average the incremental update of his proposed algorithm is done in time "proportional to the number of instances previously treated". He states that although the theoretical worst case is exponential, when one supposes there is a fixed upper bound to the number of attributes related to an object, the worst case analysis of the algorithm shows linear growth with respect to the number of objects. [15]

There exists a tool called *ConExp (Concept Explorer)* which can be used to construct and visualize a concept lattice, given a context. [6] The tool can be given a context as input and can output the corresponding concept lattice in a format readable for other applications.

4.5 Applications of FCA

FCA has been applied in many different settings and for different purposes. To explore how FCA can be applied to identify multiple archives for the MR case the several different settings for which FCA is used are discussed.

Snelting for example used FCA to study how members of a class hierarchy are used in the executable code of a set of applications by examining relationships between variables and class members, and relationships among class members. [30]

Eisenbarth uses FCA embedded in a semi-automatic technique for feature identification. They identify the "computational units that specifically implement a feature as well as the set of jointly or distinctly required computational units for a set of features." [8, 2]

Godin and Mili try to obtain multiple archives from a single archive and are trying to identify generalizations of (OO-)classes using FCA. As starting point they use the set of interfaces of classes. As object they use the set of classes and as attributes the set of class properties including attributes and methods. [13, 14]

The previous examples show that FCA is applicable for a wide range of purposes. The reason why FCA can be used for different settings is that the objects, attributes and the

relations are of free choice. For each purpose a different choice of the three sets is needed. For example Snelting uses variables as objects and class members as attributes.

In order to see how FCA can be applied for the MR case we discuss the work of Siff and Reps. They used FCA in a process to identify modules in legacy code. In their paper they present these three steps in the process: [28]

1. Build context where objects are functions defined in the input program and attributes are properties of those functions.
2. Construct the concept lattice from the context with a lattice construction algorithm.
3. Identify concept partitions-collections of concepts whose extents partition the set of objects. Each concept partition corresponds to a possible modularization of the input program.

Siff and Reps thus choose functions as objects and properties of functions as attributes to group together functions that have common attributes and thus are assumed to be located in the same module. They use the *concept partition* to retrieve a possible modularization. [28]

A *concept partition* is a set of concepts whose extents are non-empty and form a partition of the set of objects O, given a context (O,A,R). More formally this means that $CP = \{(X_0,Y_0)\dots(X_n,Y_n)\}$ is a concept partition iff the extents of the concepts blanket the object set and are pair wise disjoint: [26, 28]

$$\bigcup_{i=1}^{n} X_i = O \text{ and } \forall i \neq j,\ X_i \bigcap X_j = \emptyset$$

Tonella uses the same identities as objects and attributes as Siff and Reps but he uses the *concept subpartition* (CSP) to retrieve a possible modularization. Tonella proposes concept subpartitions because of an overly restrictive constraint on the concept extents to cover the complete object set. He states that for concept partitions "important information that was identified by concept analysis is lost without reason" when concepts are disregarded because they cannot be combined with other concepts. He defines that $CSP = \{(X_0,Y_0)\dots(X_n,Y_n)\}$ is a *concept subpartition* iff: [26]

$$\forall i \neq j,\ X_i \bigcap X_j = \emptyset$$

Where CSPs can be directly mapped to object partitions - that is partitions of the object set - CSPs have to be extended to the object set using the so-called partition subtraction: [26]

> The *partition subtraction* of an object subpartition SP from an object partition P gives the subpartition complementary to SP with respect to P. It can be obtained by subtracting the union of the sets in SP from each set in P.

$$P \text{ sub } SP = \{M_k = M_i - \bigcup_{M_j \in SP} M_j | M_i \in P\}$$

P sub SP is itself a subpartition because sets in *P* are disjoint and remain such after the subtraction. The partition subtraction is used in the *subpartition extension* to obtain the object set partitioning: [26]

An object subpartition *SP* can be extended to an object partition *Q*, with reference to an original partition *P*, by the union of *SP* and the subtraction of *SP* from *P*. The empty set is not considered an element of *Q*.

$$Q = SP \bigcup (P \text{ sub } SP) - \emptyset$$

There is a remark to be made about the concept CSPs: If there is no overlap between the object sets of a set of concepts, the number of CSPs resulting from this set of concepts will grow enormously: The number of CSPs are then the number of ways to partition a set of n elements into k nonempty subsets. (This number is given by the Stirling numbers of the second kind.)

The previously presented applications of FCA, the process presented by Siff and Reps, and the definitions of concept partition and concept subpartition can be used to apply FCA to the MR case. [28, 26]

4.6 Application to the MR case

In this section is discussed how FCA can be applied for the MR case. In section 4.6.1 the approach of applying FCA to the MR case is discussed, in section 4.6.2 the 'features' in the context that is used are more deeply elaborated and in section 4.6.3 scalability issues concerning the presented approach are discussed. Finally in section 4.6.4 the notion of 'concept quality' is defined to aid the analysis on the archive.

4.6.1 Approach

To obtain multiple independent archives for the MR case using FCA the first step is to build a suitable context. For the object set in the context initially the set of files in the MR archive was chosen. The reason is that a file can be seen as a good abstraction of functionality; Choi states that functions in a file are semantically related and form a "cohesive logical module". [5]

However early experiments with formal concept analysis with files gave such amount of scalability issues that one step higher in the level of abstraction was chosen from the archive of the MR case. This level of abstraction results in a set of *building blocks*.

The choice of taking building blocks as the set of objects is not only the most logical with respect to the level of abstraction, also domain experts of PMS indicate that building blocks are designed to encapsulate particular functionality within the archive.

The attributes have to be chosen in such way that building blocks that are highly related to each other appear in concepts of the context. The following two sets of attributes are *combined* in the context for this purpose and indicate that a building block:

1. is highly dependent on another building block, that is: uses a function or data structure of a file within a particular building block. The term 'highly dependent' is used to discriminate between the heavy use of a building block and occasional use. A threshold is used to filter relations on the degree of dependency.

2. has particular features associated with it. These particular features are: MR specificity, layering and information about what building blocks are affected during software projects. Such features are extracted from architecture overviews and project documentation or are defined by domain experts.

The reason why these two sets of attributes are identified is that the first set assumes that building blocks which are highly dependent on each other should reside in the same archive. The second set of attributes assumes that building blocks that share the same features, such as building blocks that are all very MR-specific, should be grouped in the same archive.

The first set of attributes is defined in the 'as-is' architecture; the code archive. The interdependencies of the building blocks in the architecture are extracted from the code archive using the commercial tool Sotograph and the degree of dependency is determined. [31] Sotograph determines the degree of dependency by summing up static dependencies up to the desired abstraction level. A threshold is used to be able to discriminate between a high degree of dependency and a low degree.

The second set of attributes is defined in the documentation of the architecture and is present at domain experts of the MR department. The features associated with the building blocks are discussed in more detail in section 4.6.2.

So the two sets of attributes that form the attributes of the context are a combination of the 'as-is' architecture extracted from the code archive and features extracted from the 'as-built' architecture, according to the documentation and the domain experts. In figure 4.4 a simple example of a context is shown.

Now having defined what the context is for using FCA for the MR case this paper adapts the process proposed by Siff and Reps, but adjusted for this particular purpose. [28] It follows the extract-abstract-present approach of Symphony: [32]

1. Build context where objects are building blocks defined in the software archive and attributes indicate that building blocks are highly related, using the two sets of attributes previously defined in this section.

2. Construct the concept lattice from the context with a lattice construction algorithm discussed in section 4.3.

3. Identify concept subpartitions-collections of concepts whose extents partition the set of objects using the object subpartition SP proposed by Tonella. [26] Each object subpartition SP corresponds to a possible partitioning of the archive in multiple independent archives.

MR context		attributes					
		from code archive			project documentation		
		ass_bb(1)	…	ass_bb(n)	smart_plan	64Bits	widescreen
objects	bb(1)	v					v
	bb(2)	v					v
	bb(3)			v	v		
	…						
	bb(n-1)			v		v	
	bb(n)	v		v	v		

Figure 4.4: An example context to partition the archive into multiple archives using project documentation

4.6.2 Features for FCA

As mentioned before there are two sets of attributes used for the concept analysis approach. The first set of attributes indicates whether building blocks are highly dependent on other objects. The second set of attributes defines the several features of the building blocks.

For the features two approaches have been chosen:

- Information about building blocks affected during projects
- MR-specificity and layering

These two approaches give two different results, which are evaluated further in this thesis. Both results can be used to define a final splitting of the archive, in cooperation with the systems architects.

Information extracted from project documentation

The first approach makes use of several software project documents. The documents describe which building blocks are under its scope, which means what building blocks are *expected* to be changed during the project. This scope is determined by the system's architects prior to the start of the project. The scope can consist of building blocks that are scattered through the entire archive, but because projects often are used to implement certain functionality there is a relation between the building blocks in the scope. Figure 2.2 shows an example of the scope of a fictive project.

This particular relation is used to group the building blocks together in the form of concepts after the construction of the context. The fact that this grouping possibly crosscuts the archive makes this feature interesting to use for FCA in combination with the high dependency relations between building blocks.

To give an example: given a project that implements a certain feature, named 'projectA-feature1', there is documentation at PMS that describes that 'buildingblockA', 'buildingblockB' and 'buildingblockC' are under the scope of this project, which crosscuts the code

archive with respect to the building block hierarchy. Now the feature ‘projectA-feature1’ is assigned to each of the three building blocks in the scope.

The information about scopes of projects in PMS documentation is only available for projects from the last two years. Before this time this specific information was not documented. Further, when looking at the extracted features from the documentation, it is clear that not all building blocks have one or more features designated to it. So the features do not cover the complete object set of building blocks in the context.

This has consequences for deducing concepts from a context with these features. The building block where no features have been designated to, will be grouped based on the attributes that indicate high dependency on other building blocks. This attribute however could also not be there, either because there are no dependencies from this building block to other building blocks or because the number of dependencies to another building block is below a chosen threshold. This should be kept in mind while analyzing the results.

There are around 50 documents available to extract information from, each with a scope of building blocks. So around 50 different features are assigned to the building blocks in the archive. The documents have been produced for two long-term projects.

While extracting information from the documentation some difference in detailedness was noticed. That is, some scopes of projects were defined very detailed with respect to the building blocks in the hierarchy, while others were only defined on a very high level of abstraction. In the documentation there were examples of scopes that were defined as two or three ‘subsystems’ in the archive.

For these specific cases the feature was accumulated down the building block hierarchy. For example when project documentation states that the scope of ‘projectA-feature1’ is ‘platform’, all underlying building blocks in the building block structure of ‘platform’ in the archive are given the feature ‘projectA-feature1’, including ‘platform’ itself.

So the idea of this approach is that the building blocks will be grouped based on the fact that they are related based on certain features of the software they implement or are involved with. This can be a different grouping than a grouping based on just high dependencies between the building blocks. We think it is interesting to use both types of features in the context for analysis, as a combination of the ‘as-is’ architecture and the ‘as-built’ architecture.

MR-specificity and layering

The other approach is taking into account the MR-specificity and layering of the entities in the archive. The MR-specificity can be explained like this: Some building blocks are only to be found in MR software, while others are common in all medical scanner applications or even in other applications, such as database management entities or logging functionality.

For the layering a designated scale is used for the building blocks that states whether a building block is more on the service level or more on the application/UI level. One can think of a process dispatcher on the service level and a scan-define UI at the application/UI level.

When assigning these features to each building block, building blocks that share the same characteristics of MR-specificity and layering, are grouped by deducing concepts from

a context which include these features as attributes. The underlying reason for choosing these specific features, MR-specificity and layering, is the wish of Philips Medical to evolve to a more homogeneous organization in terms of software applications. After analysis it is interesting to consider opportunities such as reusing building blocks that are common in medical scanner software from other departments or develop them together as reusable building blocks.

The MR-specificity and layering information of the building blocks in the archive is not available in documentation. Therefore to obtain these features domain experts from PMS assigned them to the building blocks. This was done using a rough scale for the MR-specificity: {*very specific, specific, neutral, non-specific, very non-specific*}. For the layering a similar scale holds starting from application/UI level to the service level. This means that the complete object set of building blocks is covered, that is each entity has a feature indicating the MR-specificity and has a feature indicating the layering. So when taking one building block into account, there are 25 combinations possible with respect to the MR-specificity and layering.

Building blocks that have certain common features - such as a group of building blocks that are 'very MR-specific' and are on the 'application/UI level' - are grouped based on these features. Of course this can be a different grouping than the grouping based on the high dependencies between building blocks. We think it is interesting to use both types of features in the context for FCA, as a combination of the 'as-built' architecture and the 'as-is' architecture.

4.6.3 Scalability issues

When using the process proposed in the previous section scalability issues come into play, because the set of building blocks in the archive are taken as the set of objects in the context. The archive consists of around 360 building blocks (not taking in account the C# coded parts of the archive) , which results in a big context with the attributes defined, with a large corresponding concept lattice.

Because identifying concept subpartitions (CSPs) from the concept lattice results in enormous amounts of CSP-collections, which have to be manually evaluated, steps must be undertaken to cope with this amount.

Leveled approach To cope with the large amount of results this paper proposes to use a *leveled approach*. The approach makes use of the hierarchical structuring of the MR archive: the archives are modularized in high level 'subsystems', which consist of multiple 'building blocks', which again are structured in a hierarchy. Figure 2.1 shows the hierarchical structuring.

By analyzing parts of the hierarchy in detail, resulting concepts from that analysis are *merged* for the next analysis. This will make the context and concept lattice of the next analysis round smaller and expected is that the resulting number of CSPs will also decrease. We will elaborate this further in this section.

First we will give a more detail detailed description of the leveled approach: By using the leveled approach some parts of the archive can be analyzed in detail, while keeping

the other parts at a high level. The results from such analysis, such as groupings of 'lower level' building blocks, can be accumulated to a next analysis where another part is analyzed in detail. These groupings are accumulated by merging the building blocks into a single fictive building block to make the context and resulting concept lattice smaller. This is repeated until all building blocks are analyzed in detail and the results are accumulated.

The attributes for each of the two approaches are assigned to the building blocks. When a part of the hierarchy of the archive is not examined in detail the attributes are accumulated to the entity that is examined globally. Figure 4.5 shows how this works. In the figure the 'platform-subsystem' in the hierarchy of the archive is analyzed globally and the others in detail. The figure shows that all the features in the lower levels in the top table of the 'platform-subsystem' are accumulated to 'platform' which can be seen in the lower table.

platform		*MR-neutral*
	basicsw	*MR-neutral, acqcontrol*
	computeros	*MR-non-specific*
	configuration	*MR-specific*
acquisition		*MR-specific*
	acqcontrol	*MR-specific, patientsupport*
	...	

platform		*MR-neutral, MR-non-specific, MR-specific, acqcontrol*
acquisition		*MR-specific*
	acqcontrol	*MR-specific, patientsupport*
	...	

Figure 4.5: Example of how features are accumulated to higher level building blocks

The analysis itself was performed using a newly developed tool - named 'Analysis Selector' - that uses the output of the analysis performed by Sotograph, which recognizes the hierarchy of the archive and relations between building blocks. Further, separate input files are given to the tool for the features, either extracted from the project planning or from MR-specificity and generality documentation.

The tool enables the selection of parts of the hierarchy to analyze in detail and enables viewing the resulting concept subpartitions and merging resulting groupings in the concept subpartitions for further analysis. After selecting what part should be analyzed in detail and what parts should not be analyzed in detail the context is created using the accumulated attributes of the context.

This context can be exported to a format that an existing tool can use as input. This tool is called 'ConExp'. [6] ConExp creates given a context the corresponding concept lattice. This can be exported again to serve as input for the newly developed tool, which can deduce concept subpartitions from the concept lattice. Figure 4.6 shows how this works.

Merging concepts Considering the number of CSPs resulting from the number of concepts, the amount of concepts taken into consideration should be kept small when calcu-

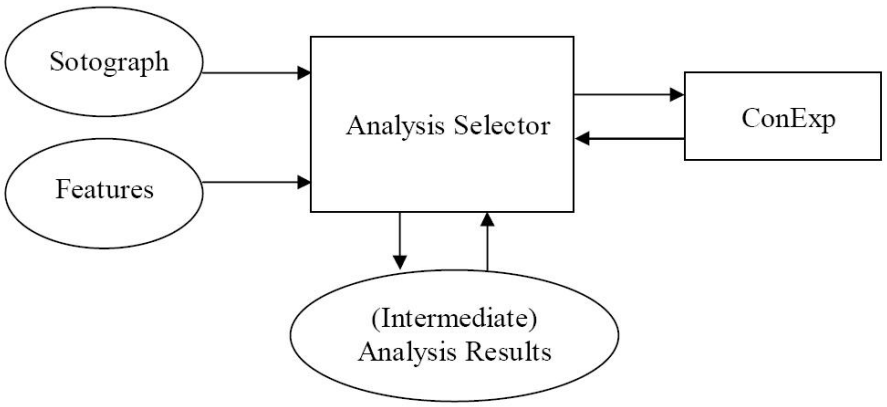

Figure 4.6: Flow of data scheme of analysis

lating the concept subpartitions. This can be accomplished by *merging* the extents of the concepts (the object sets of the concepts) resulting from the context of an analysis. We will call this *merging a concept* from now on.

When a concept is merged, the objects of that concept will be grouped into a so-called 'merge' which is a fictive object with the same attributes as the original concept. Expected is that the context for a successive analysis round is now reduced is size and a smaller amount of concepts will result from the next analysis round.

This process of merging and recalculating the context and concepts can be continued until a small number of concepts result from the defined context. From these concepts then the CSPs can be calculated.

Merging 'some' concepts raises the question which concepts to choose when merging. Picking concepts from a set of, for example, 300 concepts resulting from an analysis step is difficult. To aid in the choosing of the concepts a quality function of a concept is developed, which we will call the *concept quality*.

This quality function is developed to ease the choice of concepts to group and is not part of FCA. The quality function takes in account how 'strong' a concept is, that is based on how many attributes. Also relatively small groups of objects in a concept with relatively large groups of attributes could be ranked as a 'strong' concept. The following section elaborates the notion of 'concept quality' for this purpose.

4.6.4 Concept quality

The *concept quality* is developed as an extension for FCA to ease choosing concepts that should be merged for a next analysis round as described in section 4.6.3. In order to give each concept a value indicating the quality of the concept a quality function is needed. This function indicates how strong the grouping of the concept is.

Basically the concept quality is based on two measures:

- The relative number of attributes of a concept
- The relative number of objects of a concept

A grouping of a concept is based on the maximal set of attributes that a group of objects share. Intuitively when a small number of objects is grouped by a large number of attributes, this indicates a strong grouping of this objects and therefore is assigned a high concept quality value. Conversely, when a small number of objects share a single attribute, this intuitively can be seen as a less strong grouping than with a large number of attributes, and therefore is assigned a lower quality value.

A similar degree of quality is devised for the number of objects. Given a set of attributes, when a small number of objects share this set this can intuitively be seen as a strong grouping, on the other hand a large number of objects sharing this set as less strong grouping. So the quality function is also based on the degree of objects in a concept sharing a set of attributes. The smaller the number of objects, sharing a set of attributes, the lower the resulting quality function value is of the specific concept.

Because a degree of attributes in a concept and a degree of objects in a concept is measured for each concept, a *ceiling value* is defined. As ceiling value the maximum number of objects and attributes respectively are taken given a set of concepts. The maximum number of objects in a set of concepts is called the 'MaxObjects' and the maximum number of attributes in a set of attributes is called the 'MaxAttributes'.

Now we define the *concept quality* for a concept is as follows, the values are in the range [0,100]:

$$Quality(c) = \frac{MaxObjects - \#Objects}{MaxObjects} \times \frac{\#Attributes}{MaxAttributes} \times 100$$

Given this quality function all resulting concepts from a context are evaluated and based on the values, concepts are merged into single entities. These merges of building blocks are taken as one entity for the next round of analysis, with the purpose of decreasing the number of concepts.

4.7 Results

The analysis has been performed for the two approaches; the approach with the project documentation features and the approach with the MR specificity and layering features. Both approaches make use of the hierarchy of the archive. The results of the analysis using the two approaches are discussed in this section. Because the two approaches give two different results, the two approaches are discussed separately.

4.7.1 Project documentation features

Analysis has been performed on the complete building block structure, with no threshold imposed on the relations. This means that every static relation between building blocks is counted as a high dependency attribute in the context. This is combined with the information

extracted from the project documentation. Figure 4.7 shows the number of concepts plotted against the number of analysis rounds.

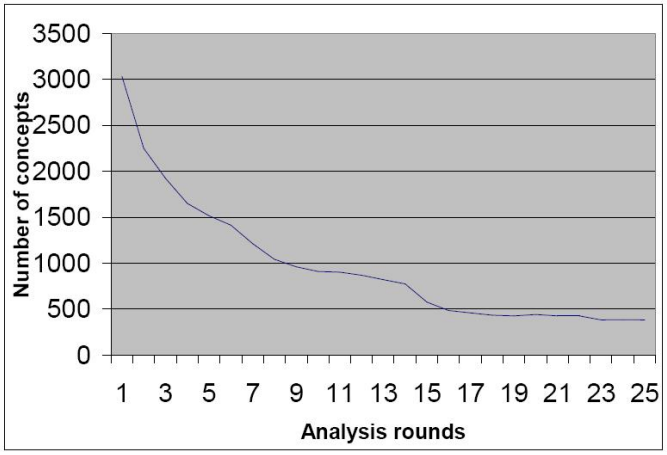

Figure 4.7: Results of analysis of FCA with features extracted from project documentation

The full context results in 3031 concepts. The number of concepts is too big to create concept subpartitions (CSPs) from. In figure 4.7 can be seen that the number of concepts decreases over the successive analysis rounds.

After 25 rounds the number of concepts has decreased to 379. However, calculating CSPs from this set of concepts resulted in more than 10 million CSPs.

When the resulting (intermediate) concepts were discussed with the domain experts, we could be seen that the objects of the concepts were mainly grouped on the high dependency attributes in the context. This can be explained by the fact that the features extracted from existing project documentation covered around 30% of the complete set of around 360 building blocks.

So when grouping objects in the defined context by concepts the main factor of grouping is defined by the dependencies. This degree of influence of the high dependency attributes can be decreased by using a threshold on the static relations, but this also means that there will be more objects in the context that have no attributes assigned to. Therefore no information is available to group these objects, so these objects will not be grouped in concepts.

For this reason we have chosen not to continue the analysis with a threshold with features extracted from project documentation and we decided not to apply FCA with these features on parts on the hierarchy.

4.7.2 MR-specificity and layering features

This section presents the results from analysis on the context with the MR-specificity and layering features.

Full building block structure The first results of the analysis with the MR-specificity and layering features are from analysis of the complete hierarchy of building blocks, without any threshold imposed on the static dependencies, used in the context. Figure 4.8 shows the results.

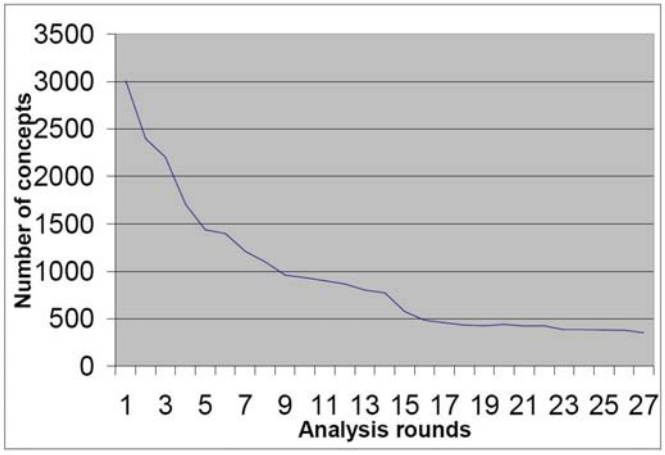

Figure 4.8: Results of analysis of FCA with MR-specificity and layering features

The full context results in 3011 concepts. Similar to the analysis on the project documentation features, this number of concepts is too big to create CSPs from. So concepts were chosen to be merged each round. Figure 4.8 shows that the number of concepts decrease over the number of analysis rounds.

After 27 rounds the number of concepts has decreased to 351. Calculating CSPs from this set of concepts resulted in more than 10 million CSPs.

The following results are from the analysis on the complete hierarchy of building blocks with the MR-specificity and layering features, but now with a threshold of 25 imposed on the static dependencies.

The full context results in 719 concepts. This number of concepts is too big to create CSPs from. Therefore concept were chosen to be merged each round. Figure 4.9 shows a decrease of the number of concepts over the successive analysis rounds.

After 6 analysis rounds the number of concepts has decreased to 378. Calculating CSPs from this set of concepts also resulted in more than 10 million CSPs.

One subsystem in detail Figure 4.10 shows the results of analysis on one subsystem - called ‘platform’ - with the MR-specificity and layering features. This analysis has no threshold imposed on the static dependencies.

There are 458 resulting concepts from the defined context. By merging concepts using the concept quality measure, after 30 analysis rounds there are 490000 CSPs resulting from the set of 95 concepts. Figure 4.11 shows the number of resulting CSPs from the last five analysis rounds.

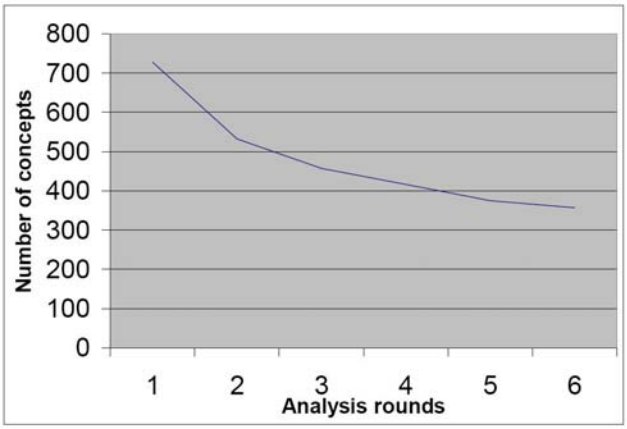

Figure 4.9: Results of analysis of FCA with MR-specificity and layering features, with imposed threshold of 25

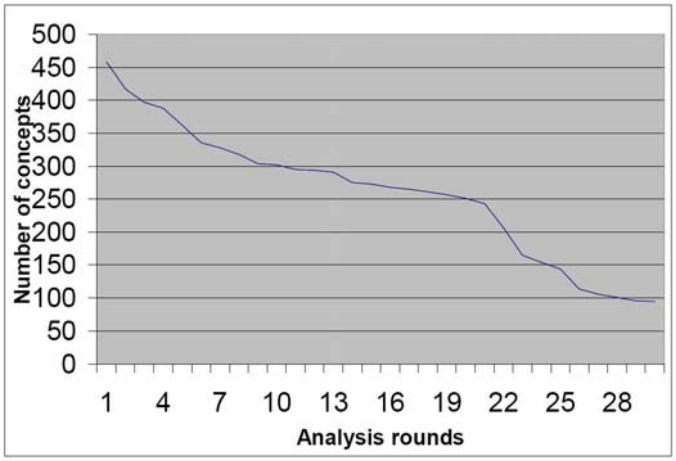

Figure 4.10: Results of analysis of FCA with MR-specificity and layering features on one subsystem

The same analysis has been performed with two different thresholds imposed on the static relations: one of 25 and one of 50. The results of this analysis can be seen in figure 4.12. The left figure shows the results with a threshold of 25 and the right figure shows the results with a threshold of 50.

The analysis with the threshold of 25 imposed starts with a context that results in 338 concepts, the analysis with the threshold of 50 imposed starts with a context that results in 287 concepts.

For the analysis with the 25 threshold after 7 rounds there was a set of 311 concepts, for the analysis with the 50 threshold there was a set of 269 concepts.

Round	Concepts	CSPs (*1000)
26	114	600
27	106	500
28	101	555
29	96	500
30	95	490

Figure 4.11: Resulting concept subpartitions from the set of concepts of the analysis on one subsystem with no threshold

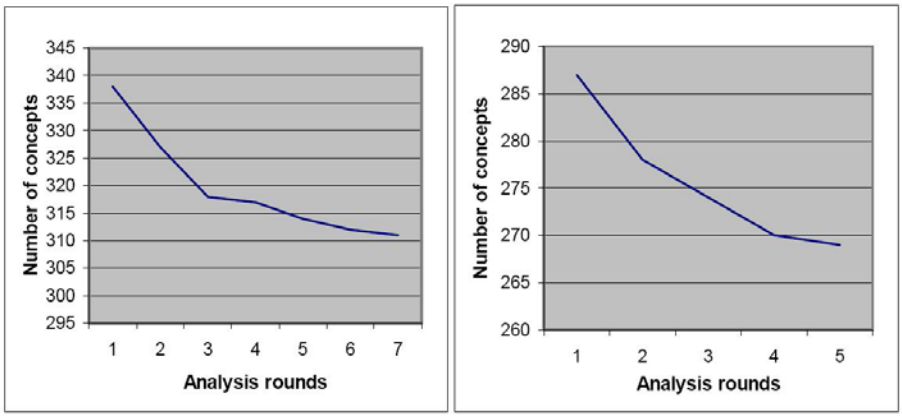

Figure 4.12: Results of analysis of FCA with MR-specificity and layering features on one subsystem, with imposed threshold of 25 (left) and 50 (right)

4.8 Evaluation

In this section the results of the previous section 4.7 are discussed and is reflected how our proposed approach works.

When we look at the results of both analyses on the complete building block structure we can see that indeed the number of concepts decrease by applying the leveled approach, aided by the concept quality values. Expected was that by decreasing the number of concepts over the several analysis rounds also the number of concept subpartitions (CSPs) would decrease.

Indeed the number of resulting CSPs decreases, if we consider both the analyses on the complete building block hierarchy. As each CSP represents a possible partitioning of the set of building blocks, all the resulting CSPs have to be evaluated by domain experts. However the number of CSPs is far too big for domain experts to evaluate.

The huge number of CSPs can be explained by the degree of overlap of the objects in a resulting set of concepts. For example, when each combination of two concepts in this set of concepts has an intersection of their objects set that is empty, the resulting number of CSPs from this set grows exponentially.

Also we observed a lot of concepts in the sets with a small number of objects. Concepts

with a small number of building blocks as the object set have a high chance that they can be combined with other concepts, as then the chance of an overlap with building blocks in the other concepts is small. More combinations of concepts result in more CSPs.

But if we consider an amount of around 100 concepts and a resulting number of around 500.000 CSPs, we do not expect to get a significant smaller amount of concept subpartitions if we choose to use different attributes in the starting context, for example other features, or a more detailed scale for the MR-specificity and layering features. We base that on the mathematical definition of the CSP, which enables small sets of concepts resulting in enormous amounts of CSPs.

So we think that the leveled approach works, considering the decreasing number of concepts. However, the approach of deducing CSPs from a set of concepts - in order to obtain partitionings of the set of building blocks - is in our opinion not applicable to a problem of the size of the MR case.

4.9 Summary

We proposed FCA in this chapter as an analysis method to obtain a splitting of the archive for the MR case. FCA provides ways to identify sensible groupings of objects that have common attributes.

For the context of FCA building blocks are taken as objects and for the attributes two sets are combined: a set of attributes indicating that building blocks are highly related and a set of attributes that represent certain features of the building blocks. These features are derived from existing project documentation and domain experts.

From this context concepts are derived, which are used to generate concept subpartitions. Each concept subpartition represents a possible partitioning of the object set of building blocks, which can serve as a basis for defining a splitting of the archive.

The process of analysis works with a leveled approach, which means that some parts of the building block hierarchy in the archive are analyzed in detail, while other parts are analyzed at a higher level. Results from a previous analysis are used for a next analysis where more parts are analyzed in detail. Selecting what parts are used for successive analysis rounds are aided by the newly defined concept quality.

After evaluating the results we see that the leveled approach, in combination with the concept quality, enables us to lower the number of resulting concepts from the context. However the resulting number of concept subpartitions from the number of concepts are enormous.

Therefore we conclude that the approach of concept subpartitions from a set of concepts is not suited for a problem of the size of the MR case.

Chapter 5

Cluster Analysis

This chapter discusses cluster analysis as an analysis method to split the software archive for the MR case. Cluster analysis can be used to analyze a system for clusters. The basics are essentially to group highly dependent objects. The group of the highly dependent objects are called *clusters*.

To give a more formal definition of the term cluster: “continuous regions of space containing a relatively high density of points, separated from other such regions by regions containing a relatively low density of points”. [9]

When applying cluster analysis three main questions need to be answered according to Wiggerts: [36]

1. What are the entities to be clustered?
2. When are two entities to be found similar?
3. What algorithm do we apply?

To discuss what cluster analysis is we follow Wiggerts by answering the three raised questions. Section 5.1 discusses the entities to be clustered, section 5.2 discusses when entities are found to be similar and should be grouped in clusters and in section 5.3 algorithms to perform the clustering are discussed. In section 5.4 applications of cluster analysis in the literature is discussed and in section 5.5 we discuss how cluster analysis can be applied to the MR case. In section 5.6 the results of the analysis are discussed and the results are evaluated in section 5.7. Finally a summary is given in section 5.8.

5.1 Entities to cluster

The entities that are to be clustered are artifacts extracted from source code. However this can be done in several ways and at different abstraction levels, depending on the desired clustering result.

Hutchens identifies potential *modules* by clustering on data-bindings between procedures. [17] Schwanke also identifies potential modules but clusters call dependencies between procedures and shared features of procedures to come to this abstraction. [27]

Another level of abstraction with respect to the entities to be clustered is presented by van Deursen. Van Deursen identifies potential *objects* by clustering highly dependent data records fields in Cobol. Van Deursen chooses to apply cluster analysis to the usage of record fields, because he assumes that record fields that are related in the implementation are also related in the application domain and therefore should reside in a object. [33]

'High-level system organizations' are identified by Mancoridis by clustering modules and the dependencies between the modules using a clustering tool called Bunch. [20, 19] Anquetil also identifies abstractions of the architecture but clusters modules based on file names. [1]

5.2 Similarity measures

When the first question is answered of what entities are to be clustered the next question that raises is when are two entities to be found similar or dependent? In other words what *similarity measure* is used to group similar entities in clusters?

As mentioned above there are different possibilities to define the similarity of entities, such as data-bindings [17] and call dependencies between entities [27]. Wiggerts divides the similarity measure found in literature in two groups: [36]

1. Relationships between entities.
2. For each entity the scores for a number of variables.

The first group is essentially based on the notion of: "the more relations between entities the more similar the entities are". The entities can be seen as nodes in a graph and the relations as weighted edges between the nodes. The relations between the entities can be dynamic or static relations, such as: inheritance/include, call or data-use relations. The several types of relations can be used together by summing up the relations of the different kinds. This type of similarity measure is applied by Mancoridis. [20, 19]

The second group of similarity measures is based on features or attributes of the entities, such as the presence of a particular construct in the source code of the entities or the degree of usage of particular constructs in the source code of the entities. Schwanke applies this type of similarity measure. [27] Also Anquetil who performs cluster analysis based on source-code file names is an example of this group. [1]

Wiggerts states that "the goal of clustering methods is to extract an existing 'natural' cluster structure". He recognizes that different clustering methods may result in different clusterings. [36] Using different similarity measures results in different clusterings of the entities. So the choice of similarity measure determines the result and in that way one should think carefully what similarity measure should be used for a particular clustering purpose.

5.3 Cluster analysis algorithms

Now the first two questions raised by Wiggerts are answered; what entities are to be clustered and how similarity between entities is determined, the next question is raised: How

are similar entities grouped into clusters? The grouping can be achieved by using different types of algorithms. Wiggerts states that a "particular algorithm may impose a structure rather than find an existing one". [36] So not only the choice of similarity measure affects the final result but also the choice of the algorithm to do the clustering with.

Wiggerts divides the existing algorithms in literature in four categories: [36]

- graph theoretical algorithms
- construction algorithms
- optimization algorithms
- hierarchical algorithms

Graph theoretic algorithms The graph theoretic algorithms work on graphs with the entities as nodes and relations between the entities as edges. The algorithms basically try to find special subgraphs in the graph. The finding of these special subgraphs in a graph is supported by graph theory. [36]

An example of the application of such algorithm is given by Botafogo. He uses a graph theoretical algorithm to cluster nodes and links in hypertext structures in more abstract structures. [4] Gibson applies a graph theoretical algorithm to identify large dense subgraphs in a massive graph; a graph showing connections between hosts on the World Wide Web. [11]

Construction algorithms Construction algorithms assign entities to clusters in one pass. An example of such construction algorithm is given by Gitman. [12] He uses fuzzy set theory to derive a clustering from a data set. Another example of such algorithm is the bisection algorithm discussed by Laszewski. [34]. The algorithm is used for dividing points or entities in a two dimensional plane into a number of partitions containing an approximately equal number of points. The membership of a point to a partition is determined by its position in the two dimensional plane.

Optimization algorithms Optimization algorithms work with a function describing the quality of a partitioning using a similarity measure. The algorithm then tries to improve the value of the quality describing function by placing entities in other clusters or by merging or creating clusters.

Mancoridis has shown that the number of possible partitions given a set of entities grows exponentially with respect to the number of entities. For example 15 entities already result in 1,382,958,545 distinct partitions. [20] This means that a naive exhaustive algorithm that calculates the quality value for each possible partition is not applicable to larger sets of entities.

Doval proposed an algorithm to deal with the exponential growth of possible partitions. [7] He uses a *genetic algorithm* and a 'modularization quality' function to cluster a graph consisting of entities as nodes and relations as edges. The algorithm starts with a random clustering and tries to come to a sub-optimal clustering by applying selection-, mutation-

and crossover-operators on a population of clusterings. The working of genetic algorithms is out of the scope of this thesis, for more details on genetic algorithms we refer to [20, 19, 7].

Doval demonstrates that the algorithm works for a small case study (20 entities). [7] Mancoridis uses a genetic algorithm in the tool *Bunch* to come to a sub-optimal clustering of given entities. [22] He presented a case study concerning 261 entities and 941 relations between the entities. The results showed that the approach worked in this case study by verification using a clustering provided by a system's expert.

The genetic algorithm in Bunch has several options that can be set by the user, which can influence the quality of the clustering. The main option is the number of successive generations of the genetic algorithm, which linearly influences the time the algorithm takes to finish.

Another optimization algorithm is the *hill-climbing* technique, a local search algorithm. This algorithm is also implemented in the Bunch tool and starts with a random partitioning of a graph consisting of entities as nodes and relations as edges. Using the 'modularization quality' function - as with the genetic algorithm - the hill-climbing algorithm start improving a random partitioning by systematically rearranging the entities "in an attempt to find an improved partition with a higher modularization quality". [22]

As a random starting solution can cause that a hill-climbing algorithm converges to a local optimum, *simulated annealing* can be used to prevent that. Simulated annealing "enables the search algorithm to accept, with some probability, a worse variation as the new solution of the current iteration". [22] This 'worse variation' can be seen as a mutation of the sub-optimal solution.

Hierarchical algorithms Hierarchical algorithms build hierarchies of clusterings in such way that "each level contains the same clusters as the first lower level except for two cluster which are joined to form one cluster". There are two types of such algorithms: [36]

- agglomerative algorithms
- divisive algorithms

Agglomerative hierarchical algorithms start with N clusters containing each one entity (N is number of entities). The algorithms then iteratively choose two clusters that are most similar - using a particular similarity measure and merge the two clusters in one cluster. Each step two clusters are merged into one cluster. After N-1 steps all entities are contained in one cluster. Now each level in the hierarchy represents a clustering. [36]

Divisive hierarchical algorithms do it the other way around. The algorithms start with one cluster containing all entities. The algorithms then choose at each step one cluster to split in two clusters. After N-1 steps there are N clusters each containing one entity. [36]

5.4 Applications of cluster analysis

Cluster analysis has been applied for many different purposes and in many settings. As mentioned in the previous sections the choice of entity, similarity measure and algorithm

determines the result of the cluster analysis. Examples of applications of cluster analysis have been given in the previous sections to illustrate these choices. This section will discuss the applications of cluster analysis with the applicability to the MR case in mind.

Wierda for example uses clustering to "group classes of an existing object-oriented system of significant size into subsystems", based on structural relations between the classes. He showed with a case study on a system consisting of around 2700 classes that cluster analysis was applicable in practice for the given problem size using a hill-climbing algorithm combined with simulated annealing. [35]

Another example in literature is the research done by Lung. Lung presents an approach to restructure a program based on clustering at the function level and applies cluster analysis on the statements in modules of around 6500 LOC. Lung states that the approach worked well, however no time measurements were included in the paper to prove the practical applicability. [18]

Other examples in literature perform cluster analysis on a far lesser amount of entities, but mainly aim at providing evidence of usefulness of cluster analysis for specific goals. [20, 19, 21, 33]

5.5 Application to the MR case

In this section we discuss how cluster analysis can be applied to the MR case. First in section 5.5.1 the approach we take is discussed, in section 5.5.2 is discussed how scalability issues are coped with and finally in section 5.5.3 is discussed how (intermediate) results are visualized and validated by domain experts.

5.5.1 Approach

In order to discuss the applicability of cluster analysis to the MR case this section tries to answer the three main questions raised by Wiggerts: [36]

- What are the entities to be clustered?
- When are two entities to be found similar?
- What algorithm do we apply?

The three questions are answered in this section. Also after answering the three questions the process of analysis is discussed.

5.5.1.1 Entities: building blocks

In order to define what the entities are, that are to be clustered the goal of the clustering should be clear and that is to identify multiple independent archives. This goal resembles the goal set by Mancoridis: Identifying high-level system organizations by clustering modules and the relations between the modules. [20, 19]

As entities we initially chose the set of files in the MR archive, as a file can be seen as a good abstraction of functionality. [5] However, the number of 15000 files gave such scalability problems in early experiments, that we chose to go one abstraction level higher. The scalability problems are caused by the fact that the number of partitions grow exponentially with the number of entities to cluster. [20] The next level of abstract in the MR case archive is the set of 360 *building blocks*.

5.5.1.2 Similarity measure: relations

The next question is when are two entities to be found similar. The choice of similarity measure affects the resulting clustering and thus should be chosen by focusing on the desired result. The desired result is obtaining independent archives from the original code archive.

To achieve independency between the archives there should be minimal *relations* between the obtained archives in order to be able to define clear interfaces between the archives. Also within an archive there should be a high degree of relations between the modules in it, because the highly related modules should be grouped into an archive.

The relations between modules are thus the most suited as similarity measure and should be extracted from the current code archive of the MR department. Because of the need to make a distinction between a strong dependency (lots or relations) and a weak one the similarity measure should measure the number of relations or degree of dependency between modules.

The choice of similarity measure is also guided by the available information at PMS, which are *static dependencies* between entities in the archive. In cooperation with the domain experts at PMS the choice was made to take these static relations as similarity measure for the cluster analysis. Other types of dependencies could also be used as relations such as dynamic dependencies and version information or file names. [**?**, 1]

As mentioned before, the MR department has a commercial tool for extracting static dependencies from the code archive, called Sotograph. The tool can handle the parts of the current code archive written in C, C++ and GOAL-C.

5.5.1.3 Algorithm: optimization

The last question that needs to be answered is what algorithm to use. The choice of the algorithm also affects the resulting clustering and therefore with the goal of this research in mind the algorithm that should be used for the MR case is an *optimization algorithm*.

The reason why is because of the goal that archives should be obtained from the code archive that can be developed independently. This means as mentioned before that there should be a low degree of relations between the archives and a high degree of relations between the modules in an archive. These two statements can be translated in a function that evaluates a clustering and can be used to obtain the optimal clustering based on this function.

We have chosen for a *genetic algorithm* to cope with the exponential growth of the number of possible partitions. We assume that a genetic algorithm is less likely to get stuck in local optima than the hill-climbing algorithm.

The reason for this assumption is the fact that with a genetic algorithm mutation is embedded in the process of selecting what partitions are continued with, which means that the algorithm can make jumps of varying size in the search space. Also in the algorithm sub-optimal solutions are recombined (cross-over function), which allows large structured jumps in the search space of the problem.

Hill-climbing in combination with simulated annealing can also prevent getting stuck at local optima, because simulated annealing is basically a (often small) mutation on the sub-optimal solution. However no cross-over is provided. Further in this section the theory needed for the optimization algorithm is discussed.

Intra-Connectivity To come to a function that can be used to evaluate a clustering first we need to translate the notion of the low degree of relations between archives. Therefore we adopt the definition of *Intra-Connectivity (A)* from Mancoridis. [20] In his paper he describes intra-connectivity as the "measurement of connectivity between the components that are grouped together in the same cluster". The components or modules that are grouped together in the same cluster should be highly dependent. That is why Mancoridis defines the intra-connectivity as follows:

$$A_i = \frac{\mu_i}{N_i^2}$$

The intra-connectivity measurement A_i of cluster i consists of N_i components and μ_i intra-edge dependencies. The measurement A_i is a fraction of the maximum number of intra-edge dependencies, because μ_i is divided by N_i^2. Here N_i^2 is the maximum number of intra-edge dependencies that can exist between N_i modules.

The measurement A_i has values in the range $[0,1]$. It has the value 0 when there are no intra-edge dependencies and the value 1 if cluster i consists of components that are fully connected which each other (including themselves).

Inter-Connectivity To translate the notion of a high degree of relation between modules in an archive the definition of *Inter-Connectivity (E)* from Mancoridis is adapted. [20]. Mancoridis describes inter-connectivity as the "measurement of the connectivity between two distinct clusters. As mentioned before the degree of dependency between the clusters should be minimized. To be able to minimize this automatically Mancoridis defines the inter-connectivity as follows:

$$E_{i,j} = \begin{cases} 0 & \text{if } i = j \\ \frac{\varepsilon_{i,j}}{2N_iN_j} & \text{if } i \neq j \end{cases}$$

The inter-connectivity measurement $E_{i,j}$ between clusters i and j consists of N_i and N_j components respectively, with $\varepsilon_{i,j}$ inter-edge dependencies. The measurement $E_{i,j}$ is a fraction of the maximum number of inter-edge dependencies possible between two clusters which is $2N_iN_j$.

The measurement $E_{i,j}$ also has values in the range [0, 1]. It has the value 0 when there are no inter-edge dependencies between the modules of the two clusters and the value 1

when all the modules in the two clusters are dependent on all the other modules in the other clusters.

Modularization Quality Now the *Modularization Quality (MQ)* measurement can be defined in order to be able to give a value for how good a particular partitioning is. This is done using the following formula defined by Mancoridis and integrating the two previous measures: 'intra-connectivity' and 'inter-connectivity'. [20]

$$MQ = \begin{cases} \frac{1}{k}\sum_{i=1}^{k} A_i - \frac{1}{\frac{k(k-1)}{2}}\sum_{i,j=1}^{k} E_{i,j} & \text{if } k > 1 \\ A_1 & \text{if } k = 1 \end{cases}$$

The formula rewards the creation of highly dependent clusters while penalizing the creation of too many inter-edges between the clusters. Using this formula, clustered partitions can be evaluated and compared. The tool *Bunch*, which uses this formula, is able to automatically obtain (sub-)optimal clustered partitions. [19]

Bunch Bunch uses as input a so called *Module Dependency Graph* (MDG). Formally, $MDG = (M,R)$ is a graph where M is the set of entities and $R \subseteq M \times M$ is the set of ordered pairs $\langle u,v \rangle$ that represent the source-level dependencies between the entities u and v. [19] This means that the chosen entities (files) and similarity measure (relation) can be mapped to such graph.

Mitchell and Mancoridis developed an improvement of the MQ measurement previously discussed, which also supports weighted edges between entities. The improved MQ measurement has been integrated into the Bunch tool. [23]

Also Bunch provides a feature called 'user-directed clustering'. When this feature is enabled, the user is able to "cluster some modules manually, using their knowledge of the system design while taking advantage of the automatic clustering capabilities of Bunch to organize the remaining modules". [19]

5.5.1.4 Process of analysis

Having defined what the entities to be clustered are, what the similarity measure is and what algorithm to use, the next point of discussion is the process of analysis. As the algorithm proposed is likely to produce suboptimal partitions there should be a way to iteratively come to a satisfying solution.

The genetic algorithm proposed by Doval starts with a random initial population, that is a random partitioning. After that genetic operators are applied to this population for some maximum number of times. Then, the fittest population is shown, i.e. the population that has the highest Modularization Quality from all solutions generated by the algorithm. [7]

The main problem with this algorithm is the fact that it starts with a random partitioning which can be a very bad partitioning. When as a starting point a bad partitioning is taken, the algorithm is likely to find a suboptimal solution, which would imply a bad partitioning of the multiple archives solution, originating from the original single archive.

Therefore a good starting point is necessary. This paper presents a process to iteratively improve the starting point of the genetic algorithm by using the suboptimal solution of the algorithm and by reviewing the solution with domain experts, such as architects and developers of the MR department. It follows the extract-abstract-present approach of Symphony: [32]

1. Obtain the Module Dependency Graph of the code archive using the commercial tool Sotograph.
2. Use the genetic algorithm provided by the tool Bunch to obtain a clustering of the MDG.
3. Review the results with domain experts by visualizing the results.
4. Use newly obtained knowledge of the system as starting solution for the genetic algorithm provided by Bunch. This knowledge is acquired in the previous step where domain experts review the results.
5. Repeat steps 2 till 4 until satisfying results are obtained, validated by the domain experts and the value of the MQ function.

When new architectural knowledge about the clusters is obtained and the search space of the algorithm is made smaller, the optimization value MQ of the initial solution is improved, meaning that the genetic algorithm can obtain a better optimization.

As mentioned above the process is repeated until satisfying results are obtained. Whether the results are satisfying is determined by domain experts and this can be seen as validation of the process. This validation is difficult when results are not visualized. Therefore a plug-in for the tool Sotograph is written. More details on the visualization can be found in section 5.5.3

5.5.2 Scalability issues

When applying the process proposed in the previous section obtained knowledge of the system is used to come to a better clustering solution. However there are scalability issues concerning this approach.

The search space for the analysis is extremely large, as the number of possible clusterings grows exponentially with the number of building blocks. As such, a first run clustering of the building blocks will probably result in a clustering from which domain experts cannot extract usable information.

Therefore we propose to use a *leveled approach*, already discussed in section 4.6.3 for FCA. The leveled approach makes use of the hierarchical structure of the archive. The archive is composed of multiple ‘subsystems’, which consist of multiple ‘building blocks’. Figure 2.1 shows the hierarchical structuring.

Certain parts of the archive can be analyzed in detail while other parts of the archive are analyzed at a high level. Results from analysis of one part of the archive can be used for a next iteration of the analysis.

For example one subsystem can be analyzed in detail, while other subsystem are analyzed globally. When a subsystem is analyzed globally, the relations from all underlying building blocks in the hierarchy are accumulated to be relations from the higher level subsystem. Also relations from one building block to the underlying building blocks in the hierarchy are adjusted to the higher level subsystem.

When after an analysis round some building blocks are clustered together, these building blocks can be *merged* into one entity, called a 'merge'. This merge is in fact a fictive building block. The relations are accumulated to the merge as mentioned above.

An example of a merge is shown in figure 5.1. Here the building blocks 'platform', 'basicsw', 'computeros' and 'configuration' are merged into one new merge, named 'merge'. In the figure can be seen that the relations of the named building blocks are accumulated and designated to the new building block 'merge'. Also the relations *to* the building blocks are changed to relations to the merge.

platform		
	basicsw	acquisition 5
	computeros	acquisition 5
	configuration	acqcontrol 2
acquisition		
	acqcontrol	basicsw 5
	…	

merge		acquistion 10, acqcontrol 2
acquisition		
	acqcontrol	merge 5
	…	

Figure 5.1: Example of a merge of a group of building blocks

The analysis itself is performed using a newly developed tool, called the 'Analysis Selector'. This tool is also used to enable the leveled approach for FCA. The tool uses the output of the static dependency analysis of Sotograph. From this output the hierarchy of the building blocks can also be extracted.

The tool enables the selection of parts of the archive to be analyzed in detail, while other parts are kept at a higher level. When the user has made his selection the necessary relation accumulation is performed and a Module Dependency Graph (MDG) is created. This MDG is used as input for the tool Bunch. This tool actually performs the cluster analysis using the built in genetic algorithm.

These results can be imported using a plug-in in the Sotograph tool to visualize the results and get more metrics on the clustering result. More information about this can be found in section 5.5.3. Domain experts can now select certain parts of the clustering to be merged for a next analysis step. The parts can be selected in the Analysis Selector, after which a next round of analysis can start. Figure 5.2 shows how this works.

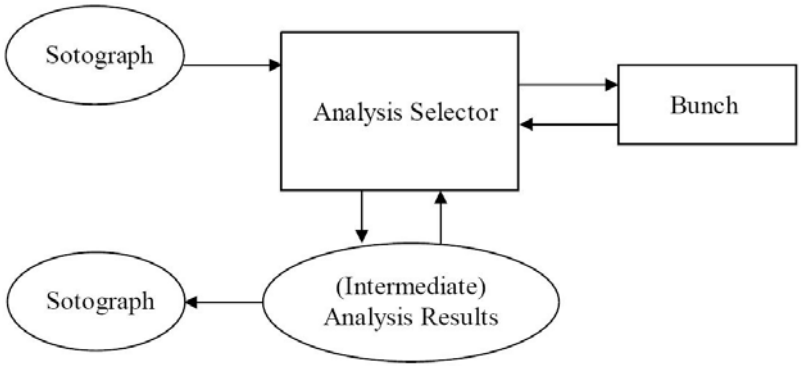

Figure 5.2: Flow of data scheme of analysis

5.5.3 Visualization and validation

The reviewing of the result of the clustering with domain experts can be aided by visualizing the results. Early results were visualized with a tool called 'dotty', which displays the clustering of the modules and the dependencies between the modules. [24] However we have chosen to use Sotograph as visualization tool, because we think that the tool has better visualization capabilities and also offers several useful metrics, such as LOC of clusters and dependency metrics. Therefore we developed a plug-in for the Sotograph tool. [31]

The results of the analyses are loaded into the Sotograph tool using the developed plug-in. Sotograph maintains a database of information extracted from static analysis of the code archive. This information includes the dependencies between archive entities, size of entities and many other metrics. The tool enables the user to visualize the clustering, get more detailed information on the dependencies between the clusters and within the clusters, see what entities are placed in a particular cluster and retrieve the sizes of the clusters.

Results of the cluster analysis are evaluated by the domain experts at PMS in the tool and building blocks are selected as merges for a next analysis round, as discussed in the previous section 5.5.2. Figure 5.2 provides an overview of our approach.

5.6 Results

This section presents the results of the several analyses on the building block hierarchy. For each analysis the scope of the analysis and the evaluation of the domain experts are discussed.

Complete building block hierarchy Cluster analysis was performed on the complete building block hierarchy. The input file given to Bunch consisted of 369 nodes (building blocks) and 2232 edges (weighted dependencies). The genetic algorithm in Bunch was executed three times. Figure 5.3 shows the results for this analysis.

Full hierarchy			
	Result	Number of clusters	Objective Function Value (MQ)
	1	84	4.034784274
	2	39	2.574595769
	3	40	3.970811187

Figure 5.3: Results of analysis on complete hierarchy

As can be seen in figure 5.3 the MQ values do not differ significantly, therefore each of the three results was visualized in Sotograph and was evaluated by domain experts at PMS. However the domain experts could not extract clusters of building blocks from the three clusterings to merge for a next analysis round. The reasons for that include:

- The three executions resulted in three completely different clusterings, which made them hard to compare.
- The resulting clusters differed in size (LOC) significantly, up to a factor 20000.
- Average of around 25% of the clusters contained one (low-level) building block.
- For some clusters dependencies between building blocks placed in the clusters were not evident or were not there at all.

One subsystem in detail only This analysis was performed on one subsystem only (platform). All the building blocks were analyzed in detail. The input file given to Bunch consisted of 121 building blocks as nodes and 409 weighted dependencies as edges. Figure 5.4 shows the results of three executions of the genetic algorithm in Bunch.

One subsystem detailed			
	Result	Number of clusters	Objective Function Value (MQ)
	1	12	3.906895931
	2	13	5.343310963
	3	12	4.470282759

Figure 5.4: Results of analysis on one subsystem only

All the three results were visualized in Sotograph. Domain experts have evaluated all three clusterings. Domain experts at PMS could not extract groupings of building blocks from the three clusterings for a next analysis round. The reasons for that include:

- The three executions resulted in three completely different clusterings, which made them hard to compare.
- Big difference in size of the clusters with respect to LOC, up to factor 100
- For some clusters dependencies between building blocks placed in the clusters were not evident or were not there at all.

One subsystem in detail This analysis was performed on the hierarchy with one subsystem in detail (platform), while keeping the others at the higher level. The input file given to Bunch consisted of 138 nodes (building blocks) and 655 edges (weighted dependencies). Figure 5.5 shows the results of three executions of the genetic algorithm in Bunch.

One subsystem detailed			
	Result	Number of clusters	Objective Function Value (MQ)
	1	13	3.782851141
	2	19	4.331546729
	3	15	3.921600527

Figure 5.5: Results of analysis with one subsystem in detail

All the three results were visualized in Sotograph. Domain experts have evaluated all three clusterings. Domain experts at PMS could not extract groupings of building blocks from the three clusterings for a next analysis round. The reasons for that include:

- Difficulty understanding the result of presence of multiple subsystems in one cluster. This cluster contained about 40% of the total amount of LOC of the archive.
- Big difference in size of the clusters with respect to LOC, up to factor 200
- Average of 20% of the clusters contained one (low-level) building block
- For some clusters dependencies between building blocks placed in the clusters were not evident or were not there at all.

One subsystem in detail - improved start solution This analysis was performed on the building block hierarchy with one subsystem in detail (platform), and the other subsystems at the higher level. Each of these higher level subsystems were placed in a cluster to start with, which is enabled by Bunch by giving a so called ‘user-directed’ clustering input file. The idea behind this, is that now all the building block in the platform subsystem will be divided among the other subsystems.

The input file for Bunch consisted of 138 building blocks as nodes and 655 weighted dependencies as edges and a file designated all higher level subsystems to a cluster. Figure 5.6 shows the results of three executions of the genetic algorithm in Bunch.

One subsystem detailed - improved start solution			
	Result	Number of clusters	Objective Function Value (MQ)
	1	31	3.865839028
	2	29	3.493834319
	3	27	2.750577642

Figure 5.6: Results of analysis with one subsystem in detail - improved start solution

Domain experts have evaluated the three results. The following points were observed:

- The three executions resulted in three completely different clusterings, which made them hard to compare.
- Average of 20% of the clusters contained one (low-level) building block
- For some clusters dependencies between building blocks placed in the clusters were not evident or were not there at all.
- The third clustering has a nice distribution of the subsystems over the clusters; there are no multiple subsystems present in a cluster

Having evaluated the three results, decided was with the domain experts that the results from the third clustering are taken for a next analysis round. Decided was to take all clusters containing more than 3 building blocks as a merge for the next analysis. The remaining clusters were seen by the domain experts as not logical and therefore the building blocks contained in these clusters are used as single entities for the second analysis round. Figure 5.7 shows the results of one execution of the analysis. (We have chosen to perform one execution, because of lack of time to run and review three executions for each setting at the end of the project.)

One subsystem detailed - improved start solution, 2nd round			
	Result	Number of clusters	Objective Function Value (MQ)
	1	12	2.132841135

Figure 5.7: Results of analysis with one subsystem in detail - improved start solution, round 2

The input file for Bunch consisted of 42 nodes and 226 edges. When visualizing the results of the analysis in Sotograph the domain experts observed that there was one huge cluster containing around 40 % of the lines of code of the total archive. This is not desirable for PMS and therefore was decided not to continue with the analysis of one subsystem, while keeping the other subsystems at the highest level.

Two predefined large clusters - improved start solution This analysis was performed on the building block hierarchy, having 2 predefined large clusters as start solution. This start solution with the 2 predefined clusters is achieved by merging the building blocks in the clusters before the analysis is executed. The clusters were defined by domain experts: one cluster contained mainly image handling software and the other contained mainly scanning software. The first cluster contains 2 subsystems from the building block hierarchy and the latter cluster contains 11 subsystems from the building block hierarchy. The remaining 2 subsystems were analyzed in detail.

As a start solution there is already a ‘bipartition’ of the archive. The idea behind the analysis is to see how the remaining building blocks are divided among the two partitions, or see how the remaining building blocks are grouped in new partitions. Figure 5.8 shows the results of one execution of the genetic algorithm.

Bipartition start solution			
	Result	Number of clusters	Objective Function Value (MQ)
	1	29	2.022904392

Figure 5.8: Results of analysis with two predefined large clusters

The input file given to Bunch consisted of 129 building blocks as nodes and 485 weighted dependencies as edges. Domain experts observed the following items:

- A few building blocks were divided among the two predefined clusters.
- The majority of the building blocks were clustered in 27 clusters
- For some of the 27 clusters dependencies between building blocks placed in the clusters were not evident or were not there at all.

With the domain experts was decided to continue with the analysis, by merging the building blocks that were clustered with the two predefined clusters. The remaining building blocks (in the 27 other clusters) were not merged and are used as single entity for the next analysis round. Figure 5.9 shows the results of one analysis with the genetic algorithm in Bunch.

Bipartition start solution, 2nd round			
	Result	Number of clusters	Objective Function Value (MQ)
	1	20	3.069774546

Figure 5.9: Results of analysis with two predefined large clusters, round 2

The input file given to Bunch consisted of 109 building blocks as nodes and 385 weighted dependencies as edges. Domain experts observed the following items:

- A few building blocks were divided among the two predefined clusters.
- The majority of the building blocks were clustered in 18 clusters
- For some of the 18 clusters dependencies between building blocks placed in the clusters were not evident or were not there at all.

Because the building blocks that were clustered with the two predefined clusters had very low dependencies or even no dependencies on the other building blocks in the clusters, we decided that continuing with this analysis - by merging the added building blocks to the two predefined clusters - was not useful. Therefore we decided not to continue with the analysis on the two predefined clusters.

5.7 Evaluation

In this section the results of the previous section 5.6 are discussed and is reflected how our proposed approach works.

Cluster analysis on the building block hierarchy did not result in a satisfying clustering, that could serve as basis for the splitting of the archive. The main reason for that is that it is difficult for the domain experts to get merges from an analysis round when the results are not ‘acceptable’.

The reasons why the intermediate results were not acceptable for the domain experts are the following:

- Different executions of the algorithm result in different clusterings, which makes it hard to compare the results
- Multiple clusters contain one building block
- Multiple clusters contain building blocks that are not related

This means that the proposed process with the leveled approach only works if results from previous analysis rounds are at least to some degree acceptable, that is domain experts can filter out meaningful clusters.

The fact that the intermediate results could not be used by the domain experts in most cases is also caused by the genetic algorithm of Bunch. The algorithm produces results with clusters that highly differ in size (LOC). The system’s architects found that this is not desirable for the MR case. ’Cluster size equality’ is not a requirement made explicit in this thesis, however this requirement appeared to be an important criterion for evaluating the results by the domain experts.

The genetic algorithm also in some cases produced results with a cluster that contained up to 40% of the size of the archive. This is also not desirable for the MR case.

5.8 Summary

This chapter proposed cluster analysis as a method to obtain multiple independent archives from the original code archive of the MR case. As entities for analysis building blocks are taken, static dependencies as similarity measure and a genetic algorithm to execute the actual analysis.

Analysis has been performed using the leveled approach on several parts of the archive. This means that some parts of the archive are analyzed in detail, while other parts are analyzed at a higher level. By using the leveled approach results on one part of the archive can be used for a next analysis, where other parts are analyzed in detail.

Domain experts evaluated the results by visualizing them in Sotograph. They could not filter groupings of building blocks from clusterings that could serve as ‘merges’ for a next analysis because of the following reasons:

- Different executions of the algorithm result in different clusterings, which makes it hard to compare the results

- Multiple clusters contain one building block
- Multiple clusters contain building blocks that are not related

The intermediate results of the analysis were not ‘acceptable’ for the domain experts, and therefore they could not filter ‘merges’ for successive analysis round. This means that no splitting of the archive could be obtained using the proposed approach.

Chapter 6

Conclusions and future work

This thesis aims at ways to split the software archive of MR into multiple archives that can be developed independently. The framework of Symphony is used to come to a solution of this problem. Two analysis methods are used within the framework: Formal Concept Analysis (FCA) and cluster analysis.

In this thesis we have made the following contributions:

- We discussed how FCA and cluster analysis can be applied to a large-scale, non-trivial industrial software archive. In literature we could not find cases with comparable goals and scale.
- We presented the *leveled approach*, which makes use of the existing hierarchy in the software archive, in order to cope with scalability issues that come with applying FCA and cluster analysis on a large-scale software archive.
- We presented the *concept quality*, a measure that aids in choosing concepts from a set of concepts.

Conclusions and future work for FCA are discussed in section 6.1 and for cluster analysis in section 6.2.

6.1 Formal Concept Analysis

The first analysis method that we have presented is FCA. FCA provides ways to identify sensible groupings of object that have common attributes. For the context of FCA we defined building blocks as objects and for the attributes two sets are combined: a set of attributes indicating that building blocks are highly dependent on each other - using a threshold - and a set of attributes that represent certain features of the building blocks. These features are derived from existing project documentation and domain experts.

From this context we derived concepts, which are used to generate concept subpartitions (CSPs). Each CSP represents a possible partitioning of the object set of building blocks, which can serve as a basis of defining a splitting of the archive.

The process of analysis works with a leveled approach, which means that some parts of the building block hierarchy in the archive are analyzed in detail, while other parts are analyzed at a higher level. Results from previous analyses are used for successive analysis rounds, where more parts are analyzed in detail. Selecting what parts are used for the successive analysis rounds are aided by the newly defined concept quality.

When looking at the results with the domain experts at PMS, the idea of applying the leveled approach in combination with the concept quality works well: The number of concepts decrease significantly. However the resulting number of CSPs is enormous. Therefore we think that FCA in this setting - obtaining CSPs from the set of concepts, given a context of reasonable size - is practically not applicable.

FCA is in our opinion very suited for recognizing certain patterns or groupings of objects based on attributes, but not suited for an analysis that should result in a precise non-overlapping partitioning of the object set.

6.2 Cluster analysis

The second analysis method is cluster analysis. Cluster analysis is used to group highly dependent objects in clusters. We have chosen building blocks as entities to cluster. Static dependencies are used as similarity measure and we use a genetic algorithm to evaluate clusterings on a clustering quality measure.

The process of analysis works with a leveled approach, meaning that some parts of the code archive can be analyzed in detail, while other parts of the archive are analyzed at a higher level. This enables us to use results from previous analyses for successive analyses, by merging clusters of building blocks that are validated by domain experts.

However the domain experts could not filter groupings of building blocks from clusterings that could serve as ‘merges’ for successive analysis rounds. One reason for that is the fact that different executions of the genetic algorithm resulted in different clusterings, which made it hard for the domain experts to compare the results.

The other reason is the fact that some clusters in the resulting clusterings were not acceptable for the domain experts: the size of resulting clusters differed significantly and some clusters contained only one building block, or contained building blocks that were hardly or not related.

This means that no splitting of the archive could be obtained using the proposed approach. However we do think that using this leveled approach is a good way to cope with the scalability issues, with the remark that intermediate results should be ‘good’ enough to use for successive analysis rounds.

These ‘good’ intermediate results could be achieved by implementing some implicit requirements the domain experts pose on clusterings in the clustering algorithm. For example parameters that can achieve equally sized clusters, or parameters that can set a range of the number of desired clusters. This might help to input domain knowledge into the analysis process, which could narrow the search space of the algorithm.

Bibliography

[1] N. Anquetil and T. C. Lethbridge. Recovering software architecture from the names of source files. *Journal of Software Maintenance*, 11(3):201–221, 1999.

[2] D. Bojic, T. Eisenbarth, R. Koschke, D. Simon, and D. Velasevic. Addendum to ”locating features in source code’. *IEEE Trans. Softw. Eng.*, 30(2):140, 2004.

[3] J. P. Bordat. Calcul pratique du treillis de galois d’une correspondance. *Mathematiques et Sciences Humaines*, 96:31–47, 1986.

[4] R. A. Botafogo and B. Shneiderman. Identifying aggregates in hypertext structures. In *HYPERTEXT ’91: Proceedings of the third annual ACM conference on Hypertext*, pages 63–74, New York, NY, USA, 1991. ACM Press.

[5] S. C. Choi and W. Scacchi. Extracting and restructuring the design of large systems. *IEEE Software*, 7(1):66–71, 1990.

[6] ConExp. http://sourceforge.net/projects/conexp, 2007.

[7] D. Doval, S. Mancoridis, and B. S. Mitchell. Automatic clustering of software systems using a genetic algorithm. In *STEP ’99: Proceedings of the Software Technology and Engineering Practice*, page 73, Washington, DC, USA, 1999. IEEE Computer Society.

[8] T. Eisenbarth, R. Koschke, and D. Simon. Locating features in source code. *IEEE Transactions on Software Engineering*, 29(3):210–224, 2003.

[9] B. Everitt. *Cluster Analysis*. Heineman Educational Books, London, 1974.

[10] B. Ganter and R. Wille. *Formal Concept Analysis: Mathematical Foundations*. Springer-Verlag New York, Inc., Secaucus, NY, USA, 1997.

[11] D. Gibson, R. Kumar, and A. Tomkins. Discovering large dense subgraphs in massive graphs. In *VLDB ’05: Proceedings of the 31st international conference on Very large data bases*, pages 721–732. VLDB Endowment, 2005.

[12] I. Gitman and M.D. Levine. An algorithm for detecting unimodal fuzzy sets and its application as a clustering technique. *IEEE Transactions on Computers*, C-19:583–593, 1970.

[13] R. Godin and H. Mili. Building and Maintaining Analysis-Level Class Hierarchies Using Galois Lattices. In *Proceedings of the OOPSLA '93 Conference on Object-oriented Programming Systems, Languages and Applications*, pages 394–410. ACM Press, 1993.

[14] R. Godin, H. Mili, G. W. Mineau, R. Missaoui, A. Arfi, and T. Chau. Design of class hierarchies based on concept (galois) lattices. *Theory and Practice of Object Systems*, 4(2):117–134, 1998.

[15] R. Godin, R. Missaoui, and H. Alaoui. Incremental concept formation algorithms based on galois (concept) lattices. *Computational Intelligence*, 11:246–267, 1995.

[16] C. Hofmeister, R.Nord, and D. Soni. *Applied software architecture*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2000.

[17] D. H. Hutchens and V. R. Basili. System structure analysis: clustering with data bindings. *IEEE Trans. Softw. Eng.*, 11(8):749–757, 1985.

[18] C. Lung, X. Xu, M. Zaman, and A. Srinivasan. Program restructuring using clustering techniques. *J. Syst. Softw.*, 79(9):1261–1279, 2006.

[19] S. Mancoridis, B. S. Mitchell, Y. Chen, and E. R. Gansner. Bunch: A clustering tool for the recovery and maintenance of software system structures. In *ICSM '99: Proceedings of the IEEE International Conference on Software Maintenance*, pages 50–59, Washington, DC, USA, 1999. IEEE Computer Society.

[20] S. Mancoridis, B. S. Mitchell, C. Rorres, Y. Chen, and E. R. Gansner. Using automatic clustering to produce high-level system organizations of source code. In *IWPC '98: Proceedings of the 6th International Workshop on Program Comprehension*, pages 45–52, Washington, DC, USA, 1998. IEEE Computer Society.

[21] B. S. Mitchell and S. Mancoridis. Comparing the decompositions produced by software clustering algorithms using similarity measurements. In *IEEE International Conference on Software Maintenance (ICSM)*, pages 744–753. IEEE Computer Society, 2001.

[22] B. S. Mitchell and S. Mancoridis. On the automatic modularization of software systems using the bunch tool. *IEEE Trans. Softw. Eng.*, 32(3):193–208, 2006.

[23] Brian S. Mitchell and Spiros Mancoridis. Using heuristic search techniques to extract design abstractions from source code. In *GECCO '02: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1375–1382, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc.

[24] Stephen C. North and Eleftherios Koutsofios. Application of graph visualization. In *Proceedings of Graphics Interface '94*, pages 235–245, Banff, Alberta, Canada, 1994.

[25] IEEE P1471-2000. Recommended practice for architectural description of software intensive systems. Technical Report IEEE P1471–2000, The Architecture Working Group of the Software Engineering Committee, Standards Department, IEEE, Piscataway, New Jersey, USA, September 2000.

[26] P.Tonella. Concept analysis for module restructuring. *IEEE Trans. Softw. Eng.*, 27(4):351–363, 2001.

[27] R. W. Schwanke. An intelligent tool for re-engineering software modularity. In *ICSE '91: Proceedings of the 13th international conference on Software engineering*, pages 83–92, Los Alamitos, CA, USA, 1991. IEEE Computer Society Press.

[28] M. Siff and T. Reps. Identifying modules via concept analysis. In *Proc. of. the Internation Conference on Software Maintenance*, pages 170–179. IEEE Computer Society Press, 1997.

[29] G. Snelting. Reengineering of configurations based on mathematical concept analysis. *ACM Transactions on Software Engineering and Methodology*, 5(2):146–189, 1996.

[30] G. Snelting and F. Tip. Reengineering class hierarchies using concept analysis. In *SIGSOFT '98/FSE-6: Proceedings of the 6th ACM SIGSOFT international symposium on Foundations of software engineering*, pages 99–110, New York, NY, USA, 1998. ACM Press.

[31] Sotograph. http://www.software-tomography.com/html/sotograph.html, 2007.

[32] A. van Deursen, C. Hofmeister, R. Koschke, L. Moonen, and C. Riva. Symphony: View-driven software architecture reconstruction. In *WICSA '04: Proceedings of the Fourth Working IEEE/IFIP Conference on Software Architecture (WICSA'04)*, pages 122–134, Washington, DC, USA, 2004. IEEE Computer Society.

[33] Arie van Deursen and Tobias Kuipers. Identifying objects using cluster and concept analysis. In *ICSE '99: Proceedings of the 21st international conference on Software engineering*, pages 246–255, Los Alamitos, CA, USA, 1999. IEEE Computer Society Press.

[34] G. von Laszewski. A Collection of Graph Partitioning Algorithms: Simulated Annealing, Simulated Tempering, Kernighan Lin, Two Optimal, Graph Reduction, Bisection. Technical Report SCCS 477, Northeast Parallel Architectures Center at Syracuse University, April 1993.

[35] Andreas Wierda, Eric Dortmans, and Lou Lou Somers. Using version information in architectural clustering - a case study. In *CSMR '06: Proceedings of the Conference on Software Maintenance and Reengineering*, pages 214–228, Washington, DC, USA, 2006. IEEE Computer Society.

[36] T. A. Wiggerts. Using clustering algorithms in legacy systems remodularization. In *WCRE '97: Proceedings of the Fourth Working Conference on Reverse Engineering (WCRE '97)*, page 33, Washington, DC, USA, 1997. IEEE Computer Society.

[37] R. Wille. *Restructuring lattice theory: an approach based on hierarchies of concepts.* Ordered sets, Reidel, Dordrecht-Boston, 1982.