TECHNISCHE
UNIVERSITÄT
WIEN

VIENNA
UNIVERSITY OF
TECHNOLOGY

MAGISTERARBEIT

ReUse Cases: Supporting Knowledge Management and Reuse with Self-Organizing Use Case Maps

ausgeführt am
Institut für Softwaretechnik und Interaktive Systeme

der Technischen Universität Wien

unter der Anleitung von
ao.univ.Prof. Dr. Andreas Rauber

durch
Christoph Becker
9952542

Gumpendorferstrasse 83-85/2/28
1060 Wien

Wien, im *März 2006*

Abstract

Organizations in today's software industry are increasingly faced with the challenge of managing information about their past, present, and future projects. The effective and efficient reuse of past knowledge, experience, and assets is one of the key success factors in the software business. To organize the huge number of documents arising during software projects, e. g. use case documents, a digital library offering content-based organization may be used. It allows the user to explore and analyze a potentially unknown library in an intuitive way.

In software reuse, *finding suitable reuse candidates* for a more or less accurately specified problem is one of the critical questions. In knowledge managment, an important issue is finding correct sources of *tacit knowledge*. Mapping the use cases of a new project to the existing collection may reveal valuable similarities that might not be uncovered by traditional information retrieval methods like key-word based search.

This thesis investigates a digital library system based on self-organizing maps (SOMs) as potential solution to these problems. SOMs are a powerful unsupervised neural network technique mapping high-dimensional input data to lower-dimensional representations while preservering the topological ordering of input data.

However, so far there have been a number of obstacles hindering this and other potential applications of SOMs to document collections. The successive steps of data extraction, indexing and feature space reduction can be very tedious and prohibitively expensive. Furthermore, the inherent structure of use cases and other texts containing several specific segments of text with different characteristic content is not taken into account yet.

To overcome these obstacles, this thesis proposes a new tool, called *SERUM – SElf oRganizing Use case Maps*, that combines and extends existing tools for indexing, feature space reduction, SOM training and map interaction to a comfortable toolset that supports the user during the complete workflow of pre-processing documents, training SOMs and exploring the resulting maps. The tool features a pattern-controlled extraction process, organization of documents according to the similarities of specific sections according to the needs of the user, sophisticated feature space reduction modules and comfortable map training.

We describe the architecture, components, and workflow, and apply the tool to organize several real-world collections of use cases obtained from industrial partners from different software branches. We point out potential benefits to a software organization and directions for future research.

Kurzfassung

Unternehmen in der heutigen Software-Industrie sehen sich zunehmend konfrontiert mit der Herausforderung, Informationen über ihre vergangenen, aktuellen und zukünftigen Projekte zu verwalten. Die effektive und effiziente Wiederverwendung von Wissen, Erfahrung und Software-Artefakten ist einer der Schlüsselfaktoren zum Erfolg. Um die gewaltige Anzahl dieser Dokumente, wie z.B. Beschreibungen von Anforderungsfällen, zu organisieren, kann eine digitale Bibliothek zum Einsatz kommen, die Dokumente basierend auf ihrem Inhalt organisiert und den Benutzern ermöglicht, eine potenziell unbekannte Sammlung auf intuitive Weise zu erforschen und analysieren.

Im Bereich der Software-Wiederverwendung ist eines der kritischen Probleme das *Auffinden geeigneter Kandidaten* für Wiederverwendung; im Bereich des Wissensmanagements ist das Finden der richtigen Quellen, die im Besitz von *stillem Wissen* sind, eine wichtige Frage. Das Abbilden von Anforderungsfällen eines neuen Projektes auf die bestehende Sammlung kann wertvolle Ähnlichkeiten aufzeigen, die mit herkömmlichen Methoden zur Informationsgewinnung wie der Suche nach Schlüsselwörtern nicht aufgedeckt werden.

Diese Arbeit untersucht *self-organizing maps* (SOMs) als eine mögliche Lösung der skizzierten Probleme. SOMs sind eine Technik aus dem Bereich der neuronalen Netze, die hochdimensionale Daten auf nieder-dimensionale Repräsentationen abbilden und dabei die topologische Ordnung der Daten beibehalten.

Momentan erschwert jedoch eine Reihe von Hindernissen diese und andere potenzielle Anwendungen von SOMs. Die notwendigen Schritte der Datenextraktion, Indizierung und *feature space reduction* sind mit den zur Verfügung stehenden Werkzeugen sehr zeitaufwändig und mühsam zu bewältigen; auch wird die inhärente Struktur von Anwendungsfällen und anderen Texten, die spezifische Text-Segmente mit verschiedenen Charakteristiken enthalten, nicht berücksichtigt.

Um diese Hindernisse zu beseitigen, stellt diese Arbeit ein neues System mit dem Namen *SERUM – SElf oRganizing Use case Maps* vor, das bestehende Module zur Textindizierung, feature space reduction sowie Training und Anzeige der SOMS kombiniert und erweitert, um den Benutzer während des gesamten Arbeitsablaufes von der Vorbereitung der Dokumente bis zur Analyse der resultierenden Karten zu unterstützen. SERUM bietet einen Muster-basierten Extraktionsprozess, die Organisation von Dokumenten nach den Ähnlichkeiten bestimmter Abschnitte, differenzierte feature space reduction und komfortables Training der SOMs.

Wir beschreiben Architektur, Komponenten und Arbeitsablauf und wenden das System an, um mehrere Sammlungen von Anforderungsfällen von Partnern aus der Software-Industrie zu organisieren. Wir zeigen potenzielle Anwendungen und Vorteile für eine Software-Organisation sowie künftige Forschungsrichtungen.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Organizations in today's software industry are increasingly faced with the challenge of managing information about their past, present, and future projects. This means both storing and organizing artifacts ranging from requirements and design documents to test cases and software source code, as well as being able to find and retrieve appropriate information when a more or less specific question arises. The effective and efficient reuse of past knowledge, experience, and assets is one of the key factors to success in the software business. The areas of software reuse and knowledge management have therefore received a lot of attention from practitioners and researchers over the last decade.

The management of information and its retrieval according to problem statements that are sometimes accurate, but often not very concrete, leads to another important area in the fields of information and computer science: digital libraries.

Digital libraries have been a prominent research topic over the last decade. In the information age we are living in, having digital objects like texts and multimedia data accesible through a computer greatly adds to the possibilities of using a library. As Levy et. al. [LM95] point out,

> The highest priority of a library, digital or otherwise, is to serve the research needs of its constituents.

The main responsibilities of a digital library are:

1. to **store digital information and organize it** according to its content, and

2. to provide users with services to find what they are looking for, i. e. **services for information discovery and retrieval**.

The problems of content-based organization and information retrieval have gained tremendous interest over the past years, and various techniques have been employed to tackle these questions.

A number of researchers started to use unsupervised neural network models, specifically the self-organizing map, to organize document collections based on their content. This approach also forms the core of the SOMLib digital library project[1].

The SOMLib system organizes text documents according to their content into a 2-dimensional grid, whereby similar documents are located near to each other on the grid. The user can thus gain an overview of the content of large, potentially unknown document collections, and find documents similar to a given one that he[2] or she would probably not find using a simple query-based retrieval method.

The self-orgainizing map has been shown to be well suited for text clustering and has been applied to document collections ranging from newspaper articles [RMD02, RM99b, RM99d] and newsgroup archives [KKLH96] to large collections of patent abstracts [KKL$^+$00].

Auer et. al.[ABRB05] applied it to collections of use case descriptions and concluded that the clustering worked very well here, too. While they used this approach for analogy-based cost estimation, it seems that other areas should profit even more from the content-based organization features of the self-organizing map.

If an organization uses a digital library approach to organize its collections of use cases with an unsupervised learning technology such as the self-organizing map, it would be able to create such a library without extensive investment, as no supervised classification schemes are employed.

Whenever a new project is undertaken and its requirements are specified with use cases, mapping these use case descriptions to the existing collection may reveal valuable similarities that might not be uncovered by traditional information retrieval methods such as keyword-based search.

Specifically, this approach is of considerable interest in two areas mentioned above: software reuse and knowledge management resp. the "Learning Software Organization".

1. In software reuse, the process of **finding suitable reuse candidates** for a given, specified problem still is one of the main problems to solve in each potential reuse instance. Leading researchers in this field plead

[1]The SOMLib project homepage can be found at `http://www.ifs.tuwien.ac.at/~andi/somlib/index.html`

[2]For sake of readability, we stick to the male pronoun throughout this work. Of course, we mean both genders.

to rely on artifacts on the level of the problem space for the retrieval of reuse candidates – as opposed to the often dominating usage of artifacts in the solution space coming from later stages, such as source code documentation. For example, Jacobson[JGJ97] writes

> If we delay considering reuse to the design or implementation stage, we may find our design is incompatible with the features offered ...

Use cases are one of the leading methodologies in software requirements engineering and are being applied in countless organizations throughout all software industries. Thus the potential data base for experiments in this area is very large.

2. In knowledge management, a critical problem is pointing a user to the correct source of information for a specific problem, which often means referring him to a colleague holding valuable information in form of **tacit knowledge**. With use case descriptions pointing to their responsible authors, this can be achieved with minimum effort.

Using a digital library of use case descriptions for software reuse and knowledge management thus is a promising approach, but to investigate its possible benefits and limitations in these areas, detailed experiments are needed.

To cluster use cases in a self-organizing map, the descriptions need to be extracted from larger requirements documents, as most organizations do not rely on dedicated requirements management tools, but simply on text-processing applications, for specifying the requirements for their projects.

Ideally, a user could take a document containing a number of texts that he or she wants to be organized in a map, feed it into the system, and be presented with a map showing the extracted, indexed, and organized text documents.

But this scenario is not yet reality. Figure 1.1 shows the steps that have to be conducted to use the SOMLib toolsuite. So far, there are several obstacles hindering the wider accepted usage of the SOM.

- First, the **extraction process is tedious**. To apply the SOM algorithm, feature extraction has to be done in the form of text indexing similar to that done inside a search engine, which transforms documents into a numerical representation. This process usually works on single files. But relevant document passages in real life are contained in larger documents, which have to be split up first in order to allow

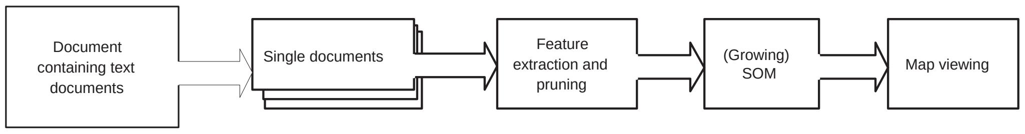

Document containing text documents
Single documents
Feature extraction and pruning
(Growing) SOM
Map viewing

Figure 1.1: Steps needed to organize texts contained in a documents with a self-orgaizing map.

text indexing. So far, this has to be done in a manual preprocessing step that can take quite a long time and may be prohibiting in terms of effort.

- Second, tool support **automating the workflow** of extracting, indexing, and training is yet quite poor. The user has to manually handle the input and output files and feed the input files of preceding steps to successive tools.
- Third, tool support allowing the user to **view the content of the original documents** is not very user-friendly.
- Moreover, current approaches do not take into account the structured nature of text documents like use cases, which consist of several text sections with specific characterstics and relations across the single documents; clustering documents according to the content of specific sections, which could lead to a better insight into specific aspects of document collections, can only be done by manually partitioning the text.

To overcome these problems and thus support the application of SOMs to software reuse and the Learning Software Organization, this thesis focuses on tool enhancement for the SOMLib digital library system to facilitate an automated workflow ranging from the extraction and indexing to the map interaction stages.

Goal is the **further development, enhancement and integration** of the existing components of the SOM Toolbox and the SOMLib digital library to a tool that allows the user in a comfortable way

- to **import** in a way **as automated as possible collections of segmented documents containing short texts** (in particular requirements specifications containing use case descriptions),
- to manage these document collections and **organize them according to configurable criteria in SOMs**. The resulting maps should be saved, and an extended SOMviewer shall present them in a convenient and helpful way and thus **support the analysis process**.

For use case documents, the user defines templates with specified parts and keywords. The system imports use cases according to these patterns into the database, trains SOMs according to the user's needs and presents them to the user for browsing and exploration, which includes viewing the text of the original use cases.

The resulting tool is being developed under the working title "**SERUM – SElf oRganizing Use case Maps**".

The remainder of this thesis is structured as follows.

- Chapter 2 explains the self-organizing map models and tools underlying the tool suite to be developed, lines out relations to digital libraries and text mining and introduces use cases and the application areas discussed later, namely software reuse and knowledge management respectively the "Learning Software Organization".
- A detailed description of the requirements, design and the components of the SERUM toolsuite follows in chapter 3.
- In chapter 4, we describe the application of SERUM to four different collections of use cases with differing size, target domains, sources and languages. We highlight selected aspects of the feature extraction and reduction steps, and point out possible benefits to software reuse and knowledge management.
- The last chapter summarizes the work of this thesis and points out further work and research directions.

Chapter 2

Related Work

This chapter contains introductory explanations on digital libraries and text mining, as well as the self-organizing map and extensions thereof. It further outlines the SOM toolbox and the components of the SOMLib project. A short introduction to use cases leads to potential application areas of clustering use case documents, namely software reuse and knowledge management.

2.1 Digital libraries

Digital libraries have been a prominent research topic in the fields of information and computer science over the last decade, with a wealth of papers published and several series of conferences focusing on digital libraries like the European Conferences on Digital Libraries (ECDL)[1], the Joint Conferences on Digital Libraries (JCDL)[2] or the Internation Conference on Digital Libraries [3], among others.

Definitions for digital libraries vary. Lesk describes a digital library as

> a collection of information that is both digitized and organized ([Les97] as cited in [Rau00]),

and Harter defines it as

> a computerized "libary" that would supplement, add functionality, and even replace traditional libraries. [Har96]

According to Harter, the term originates from the Digital Library Initiative of the Library of Congress and the National Science Fund [NSF06]. This initiative was started in 1994 and moved on to the second phase in 1998.

[1]`www.ecdl2006.org`

[2]`http://www.jcdl.org/`

[3]`http://www.teriin.org/events/icdl/`

Digital library systems today take a vast variety of different forms. Some examples are provided below; this list is by no means meant to be complete.

- Digital archives like the **Internet Archive** dedicate themselves to archivation of content, e. g. from the web, for

  > preserving artifacts of . . . culture and heritage. [Int]

  This includes webpages, multimedia content, etc. The father figure of these archives can be seen in the ancient Library of Alexandria.

- Several efforts **initiated by conventional libraries**, often national libraries, strive to digitize content and make it available in an online form together with content already present in digital form. [BFN04]

- Other efforts include learning systems for universities, collections of lectures, or **specialized digital libraries** for scientific communities like the ACM digital library[Ass06], among many others[LGR04, iee06].

Compared to the classic paradigm of a library, modern digital libraries often present new challenges to their users who have to accustom themselves to new ways of using a library. To combine the advantages of traditional libraries, in particular the ways users interact with them, with the possibilities of digital libraries, the SOMLib digital library was conceived.

2.2 The SOMLib digital library system

This system was first presented in [Rau98], later it was elaborated in detail in [Rau00, RM99c]. In the last years, numerous experiments have led to a number of improvements and further insights[Rau03, RM03, ABRB05].

The SOMLib digital library system uses at its core an unsupervised neural network model, the self-organizing map, to organize documents according to their content. The following section describes the self-organizing map (SOM). It is followed by a discussion on the applicability of the SOM to organize document collections and an introduction to the components constituting the SOMLib digital library.

2.2.1 The Self-Organizing Map

The self-organizing map (SOM) was proposed by Prof. Kohonen in [Koh82] and later described extensively in [Koh89] and [Koh01]. It uses an unsupervised neural network to map high-dimensional input data to lower-dimensional reprentations while preserving the topological ordering of the input data as far as possible. By compressing high-dimensional information to a lower-dimensional form, usually a 2-dimensional grid structure, it visualizes this information in a way more easily accessible to human beings. It is especially popular and effective in the areas of cluster analysis, data classification and visualization of high-dimensional data.

A set of units are arranged in some topology, the most common form of which is the 2-dimensional grid. A weight vector m_i is assigned to every unit i. These weight vectors are of the same dimensions as the input data and are initialized with random values.

The input data are represented by n-dimensional vectors $x_i \in \mathbb{R}^n$ describing n features in the input space.

The training process works as follows:

1. An input vector x is selected randomly.
2. The activation of the output units is calculated according to the distance between the input vector x and the weight vector of each unit.
3. The best-matching unit c (the "winner"), i. e. the unit having the weight vector with the lowest Euclidian distance to the current input vector x, is selected.
4. The weight vectors of the "winner" c and its neighbors are modified by moving them toward the input vector x. The amount of adaption is determined by a gradually decreasing learning rate α, the size of the neighborhood depends on the neighborhood-kernel h_{ci}.
5. Steps 1-4 are repeated until the predefined stop criterion is fulfilled.

The decreasing amount of adaption as determined by the learning rate α enables first large adaption steps away from the random initialization of the weight vectors, while allowing for a fine-tuned modification of the input space representation towards the end of the training process.

As mentioned, not only a single winner unit is moved in each step, but also the units in it's vicinity as described by a Gaussian neighborhood-kernel h_{ci}. This function takes into account the Euclidian distance in the output space between a unit i and the winner c, as weel as the current time t. With r_i as the 2-dimensional vector representing the location of unit i within the grid under consideration, we get the Gaussian neighborhood kernel

$$h_{ci}(t) = e^{-\frac{{r_c - r_i}^2}{2\cdot\delta(t)^2}} \tag{2.1}$$

that scalars in the range of [0,1] and decreases during the learning process.

With α denoting the learning rate that is decreasing over the course of time, h_{ci} representing the neighborhood-kernel also varying over time, x representing the current input pattern, and m_i denoting the weight vector assigned to unit i, the learning rule is

$$m_i(t+1) = m_i(t) + \alpha(t)h_{ci}(t)[x(t) - m_i(t)] \tag{2.2}$$

Pragmatically speaking, at each step in the learning process, the weight vectors of the winner unit and the units in its vicinity are tuned towards the currently presented input vector, so that if the input vector is presented again, the activity level of the winner unit and its region will be even higher. The amount of adaption of each unit in the vicinity of the winner decreases with the distance to the winner unit.

As similar inputs are mapped onto regions close to each other, the topological ordering of input data is preserved. The decreasing neighborhood kernel allows for coarse clustering at the beginning of the learning process and fine-tuning of weight vectors towards the end, thus fostering a better topological representation of the input space.

The **quantization error** QE (also called distortion measure) is a metric for the quality of the mapping of the data onto the grid. The QE of a cell measures the dissimilarity of all input data mapped onto a particular unit by the Euclidian distance. Thus, the mean QE or MQE is the mean distance between each input vector and it's best-matching unit.

Termination criteria

The training process may be terminated by meeting a variety of different criteria. Most commonly, it is terminated

1. if a **fixed amount of iterations** has elapsed; this is a very commonly used criterion;

2. if the **quantization error** does not change any more, or

3. if a **predefined threshold** for the quantization error is reached.

Extensions of the static SOM

Despite its usefulness and effectiveness, the static structure of the classic SOM has some drawbacks [RMD02, BM93].

1. It cannot capture the inherent hierarchical structure of data;

2. The size of the map has to be specified in advance and cannot be changed later. This is a serious limitation, beause often the ideal size of the map is not known beforehand, and thus several simulations have to be run to determine the optimal sizing.

3. It is difficult to decide where the boundaries of clusters and regions are.

These restrictions led to a large amount of research (cf. [KKK98]) bringing forth several extensions that try to add dynamic growth and structural flexibility to the SOM.

- **Incremental Grid Growing** as proposed in [BM93] uses a 2-dimensional self-organizing map, starting at a size of 2x2 units. During the learning process, the grid structure is adapted in two ways. When the input data cannot be mapped to the grid with the desired granularity, units are added at the boundaries of the map. On the other hand, connections between units are added and removed according to the distances between input patterns.

- The **Growing Grid** as described in [Fri95] features a 2-dimensional rectangular ordering of units and adds rows or columns to the grid during the growth process where the input space cannot be mapped with the desired quality.

- The **Hierarchical Feature Map** is described in [Mii90] and [KO90]. It features a predefined hierarchical architecture of SOMs, thus leading to a balanced tree representing the input data. Furthermore, because the size of each map to trained is much smaller, the time needed for computation is reduced. However, it is not dynamic because the map size and the depth of the hierarchy of maps have to be defined in advance.

- The **Growing Hierarchical SOM (GHSOM)** was described in detail in [RMD02], [DMR00] and [Dit00]. It's key idea is

  > to represent the inherent hierarchical structure present in many data collections in the most accurate way. ([Dit00],p. 19)

  To achieve this, the GHSOM has a dynamic hierarchical structure of SOM layers, where data mapped onto a unit in a higher level is expanded and mapped in detail onto a new map underlying the respective unit if a specific threshold is reached. Each of the connected maps is able to grow independently of the others; the growth process within one of these maps in the hierarchy works similar to that in the growing grid, where rows and columns may be added as necessary to reach the desired granularity of the input representation.

Other extensions include Growing Cell Structures [Fri91, Fri94], the Growing SOM (GSOM) [AHS98, AHS00], and the hypercubical growing SOM [BV97].

Figure 2.1 shows the structure of various possible growth processes that can be found in the above mentioned variations of the static SOM. Cells shaded with dark grey are being selected as extension points; cells that are added during a learning iteration are indicated in light grey. Note that the hierarchical feature map is not growing dynamically during the training process; it's structure has to be predefined in advance.

For our purposes most relevant are the classic static SOM, the Growing Hierarchical SOM, and especially its flat relative, the Growing SOM. These models are readily supported by the SOMLib system, and with their 2-dimensional grid topology, they provide an intuitive visualization and excellent exploration facilities. Thus, they feature an easy-to-understand yet flexible network model.

2.2.2 Using self-orgainizing maps for clustering documents

The self-orgainizing map has been shown to be well suited for text clustering and has been successfully applied to a wide range of document collections ranging from newspaper articles and newsgroup archives to large collections of patent abstracts.

- Possibly the first to use SOMs for document collections were Lin et. al. [Lin91], who clustered a small collection of 140 documents with a static SOM.
- Kohonen et. al. used the first version of their WEBSOM tool[4] to organize messages from newsgroup archives [KKLH96].
- In a later effort, they employed the WEBSOM to organize a large collection containing of more than 6 million patent abstracts [KKL$^+$00].
- Lagus et. al.[LKK04] organized texts from the Encyclopedia Britannica with an improved version of the WEBSOM.

[4]The WEBSOM project homepage can be found at `http://websom.hut.fi/websom/`.

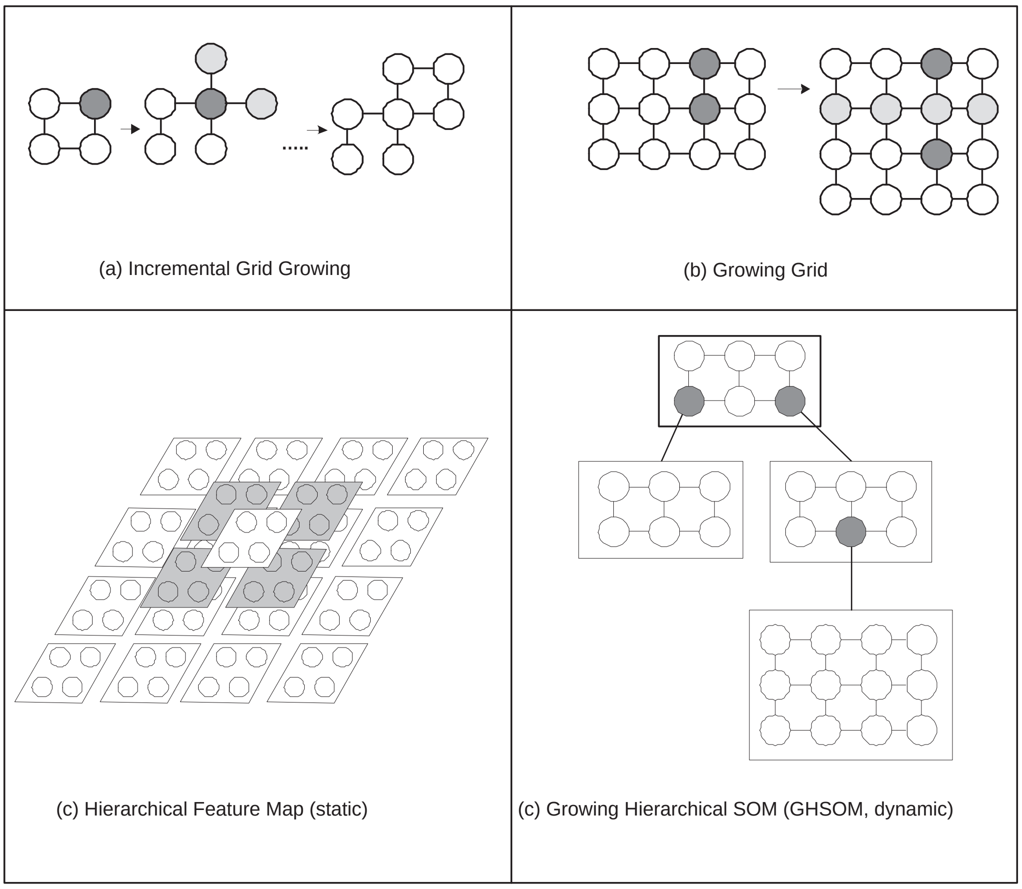

(a) Incremental Grid Growing
(b) Growing Grid
(c) Hierarchical Feature Map (static)
(c) Growing Hierarchical SOM (GHSOM, dynamic)

Figure 2.1: Examples showing the differing growth processes and hierarchy concepts in various extensions to the standard static SOM: (a) Incremental Grid Growing, (b) Growing Grid, (c) Hierarchical Feature Map, (d) Growing Hierarchical SOM (GHSOM).

- Nürnberger et. al. [NKK03] present a prototypical system for information retrieval combining keyword-based search and visualization of similarity based on content. They argue that

  > In order to visualize document collections methods are required that are able to group documents based on their similarity and furthermore visualize the similarity between discovered groups of documents. ([NKK03],p. 120)

- Rauber et. al. employed the SOMLib system to cluster a broad range of documents, among others scientific abstracts and articles from the CIA Factbook and a variety of newspapers and magazines like the TIME magazine. [RMD02, RM99b, RM99d, Rau00]

All of these different methods yielded good results in organizing documents according to the similarity of their content. As Lagus puts it,

> Experiments with the various data sets show that the method can be successfully applied to organizing both very small and very large collections, to colloquial discussions and to carefully written scientific documents... ([Lag00], p. 35)

In this work, we will focus on the SOMLib digital library, which is introduced in the next section.

2.2.3 An overview of the SOMLib sytem

The SOMLib system consists of several layers, the collaboration of which leads to a workflow that is seen by the user as depicted in Figure 2.2.

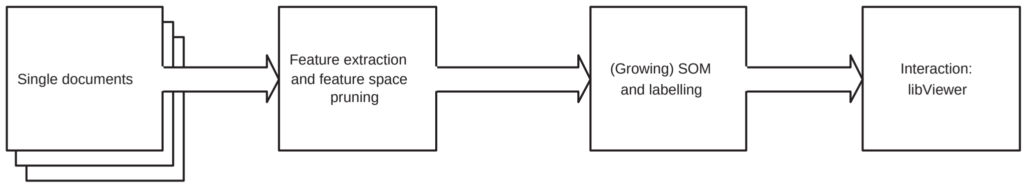

Single documents
Feature extraction and feature space pruning
(Growing) SOM and labelling
Interaction: libViewer

Figure 2.2: Overall workflow in the SOMLib digital library as seen by the user.

1. At its basis there are simple modules for **parsing text** that transform the textual representation of documents into a vector space model that can be fed into the self-organizing map.

2. This **self-organinzing map** constitutes the heart of the system. Similar documents are grouped together by feeding their vector space representation into a SOM, thus performing a cluster analysis on the documents. The SOM adapts itself to reflect the input data in the best possible way and thereby topologically preserves the inherent structure of the document collection.

3. SOMLib also includes means for the **integration of distributed libraries**, which are not of primary interest here. Please refer to [RM98, Rau98] for details.

4. More interesting for our purposes is the **LabelSOM** algorithm which automatically creates labels for clusters by analyzing term frequencies and generating lists of keywords that describe the content of documents contained in a cluster.

5. Interaction with the library is managed by the **libViewer** component, which provides a metaphor-based graphical interface to the digital library that is based on the way users interact with conventional libraries - i. e., books and folders contained in bookshelves.

To utilize a SOM for organizing a document collection, a number of preprocessing steps have to be performed. We will discuss these steps in the following. Figure 2.3 provides an overview of the detailed steps that constitute the workflow and collaboration of the components of SOMLib.

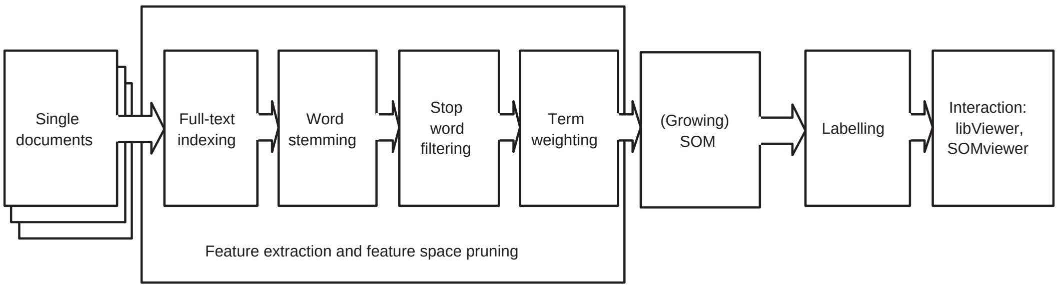


Figure 2.3: Steps involved in organizing documents with the SOMLib system in detail.

Because the SOM operates on a number of input vectors containing real numbers, **feature extraction** has to take place. Documents are preprocessed and represented in an appropriate data structure.

The starting point: Indexing text documents

A number of different approaches to text indexing can be found in information retrieval literature; some of the most important ones are *keyword indexing*, *full-text indexing* and *n-gram indexing*. As fulltext indexing is nearly completely independent of the language and can be executed automatically and efficiently without supervision (as opposed to approaches which need supervision, e. g., keyword indexing), this approach has gained a lot of interest in the information retrieval community. Since we are striving for automation and miminization of manual interaction, it can be considered the most suitable for our needs, and this is also the approach used by SOMLib.

Full-text indexing automatically creates a list of words occuring in a document, whereby the occurence of each word in the document constitutes one dimension in the feature space.

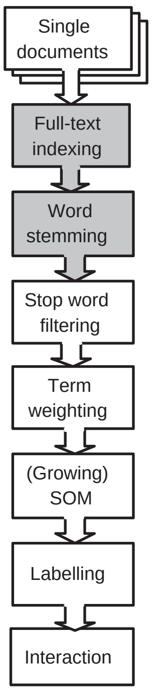


Figure 2.4: Indexing and stemming

Pruning the feature space: Word stemming

Crude indexing of every distinct word occuring in a document would lead to a situation where words like *partition*, *partitions*, and *partitioning* would be considered three distinct concepts and thus form different dimensions in the feature space. This could cause documents that contain these terms to be located far from each other, even though they contain the same semantic concepts as understood by humans. As this is certainly not what is desired, we would like to reduce words to their *stems* to unify concepts like those mentioned above. This is performed by **stemming algorithms** like the famous one introduced by Porter [Por80] that remove suffixes from words. For example, the concept mentioned above would be reduced to *partit*.

While this reduces the feature space and successfully unifies a lot of semantically equivalent terms, it also introduces language dependency.

Pruning continued: Stop word filtering

Full-text indexing generally leads to a quite high-dimensional feature space, as the number of distinct words occuring in texts of any language tends to be rather large. This applies even after stemming has been performed. Furthermore, using *all* terms contained in a document collection for clustering would cause the feature vectors of documents to be determined to a large extent by the occurence of frequent words like *the, and, of, in*, etc., that are irrelevant to the content.

Thus, we would like to remove words like these by a simple and efficient algorithm to **reduce the feature space**. There are two main concepts for doing this.

1. We can use **stop word lists** containing common words of a specified language like those mentioned above. While being efficient and quite effective, this method adds a language dependency that might not be desirable. Moreover, it may be necessary to add domain dependency, as, e. g., the term *software* may not be very distinctive for a collection of abstracts from proceedings of a conference on software engineering. This further adds a necessity for manual construction of these stop words that is not in our interest.

2. A different approach is defining **fixed limits or percentage thresholds** for the document frequency of terms. This method removes words that occur too often or too seldomly in a document collection, assuming that these words can be considered to be semantically unimportant. The SOMLib system primarily uses this approach, which has been shown to work surprisingly well. A good way is to use percentage thresholds for the maximum number of documents that a term is allowed to appear in, and a fixed limit for the minimum number of documents.[Rau00]

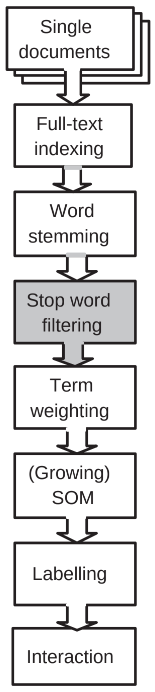


Figure 2.5: Stop word filtering

Feature representation

After the previous reduction steps, the documents have to be transformed into a representation usable for a self-organizing map. The model of choice here is the **vector space model of information retrieval** as proposed in [SWY75, SAB94].

It represents each document in terms of a vector in the feature space spanned by all distinctive words in the collection. Thus the number of dimensions is equal to the number of words occuring in the document collection (after stemming and reduction of the feature space), while the value of each component of a vector representing a document depends on the occurrence of this specific term in the document.

Different schemes for these values have been proposed. Figure 2.6 shows the position of these alternative schemes in the overall workflow.

1. The simplest scheme in this regard is **binary indexing**. This method simply assigns a boolean *true* or *false* value, resp. 1 or 0, to a component of the vector depending on whether the corresponding term occurs in the document or not. While appearing rather crude, this approach already yields quite satisfying results.

2. Nevertheless, the situation can be greatly improved by using **term weighting schemes**. We want a document to obtain a high value for a word that appears often in this document *in relation to* its total appearance in the whole document collection, thus emphasizing the *distinctiveness* of a specific term. This goal is aimed at by the prominent so-called *tfxidf* or *term frequency times inverse document frequency* scheme[SY73, Sal75]. If we consider the number of documents in a collection that a given term i occurs in as **document frequency** df_i and the number of times this term i occurs inside one document j as **term frequency** tf_{ij}, we can divide the latter by the former and obtain the weight w_{ij} of the term i for a document j as
$$w_{ij} = tf_{ij}\frac{1}{df_i} \tag{2.3}$$

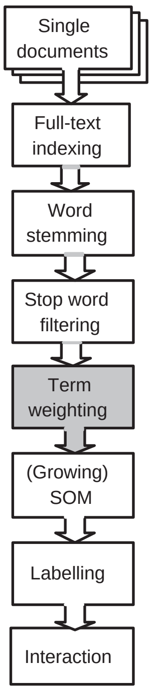


Figure 2.6: Term weighting

The *tfxidf* scheme is one of the most widely used schemes in information retrieval and also employed within the SOMLib system.

Map training

The content-based organization of documents as provided by the SOM is the core part of SOMLib. Different network models have been integrated into the SOMLib framework; the most important ones for our purposes are

- the standard static SOM with a predefined fixed size, and
- the growing SOM.

The map training outputs map description files that can be interpreted by the interaction module; furthermore, also HTML files can be generated if needed.

Labelling: The LabelSOM algorithm

Having the documents organized into topical clusters and being able to browse them in an explorative way is only part of the game; if we have to look into each document and read it before we can decide whether it is relevant or not, the representation is only half as helpful, especially if the number of documents and thus the size of the map we are exploring is large. It would be a lot easier if there would be some hints on the content of a document visible directly on the map. SOMLib addresses this issue in two ways: In the libViewer, it uses metaphors for visualizing properties of documents, as we will see in the next section. Furthermore, it uses a labelling algorithm called LabelSOM that automatically extracts keywords describing document clusters in a helpful way based on their occurence frequency.

The LabelSOM algorithm was described in [MR99, RM99e, Rau99, RM99a]. It extracts keywords that are common to the documents mapped onto a unit in the map and distinguish them from others. These keywords are displayed directly on the map.

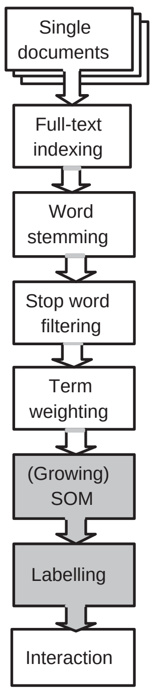
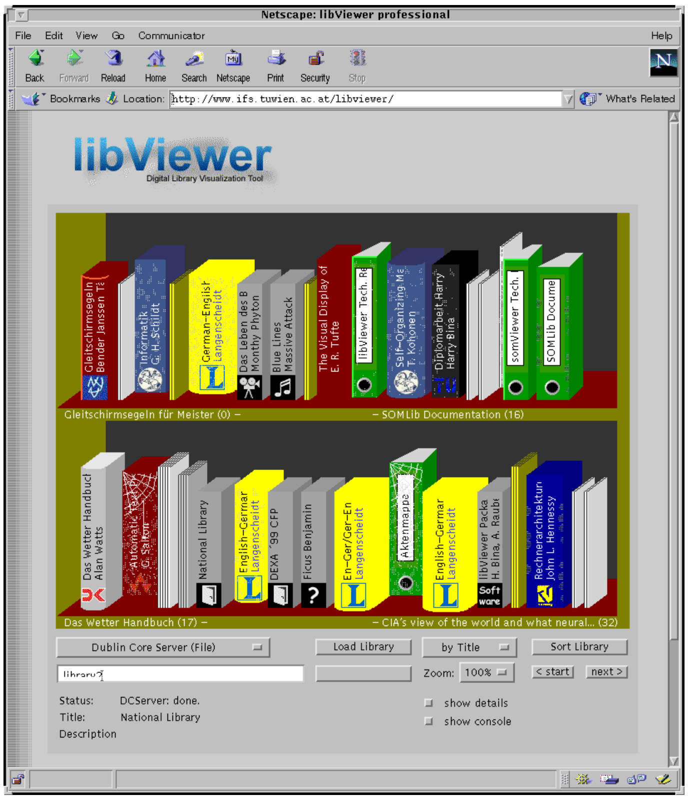


Figure 2.7: Map training and labelling

Interacting with the map

To allow users of the digital library to transfer their experience with conventional libraries to the new paradigm, a metaphor-based viewing interface to the library system was introduced: the libViewer component[RB00a, RB99].

A map is represented as a bookshelf containing books of different size, color, and shape, with labels written on the backs like titles. Furthermore, indications on the usage of books are provided like the position in the shelf, and even spiderwebs are depicted on books that have been rarely used. Figure 2.8 presents a screenshot of the libViewer interface obtained from `http://www.ifs.tuwien.ac.at/~andi/libviewer/description.html`.


Netscape: libViewer professional
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Stop
Bookmarks Location: http://www.ifs.tuwien.ac.at/libviewer/
What's Related
libViewer
Digital Library Visualization Tool
Gleitschirmsegeln für Meister (0) –
– SOMLib Documentation (16)
Das Wetter Handbuch (17) –
– CIA's view of the world and what neural... (32)
Dublin Core Server (File)
Load Library
by Title
Sort Library
Zoom: 100%
< start
next >
show details
show console

Status: DCServer: done.
Title: National Library
Description

Figure 2.8: libViewer interface in action.

Although an innovative approach, this interface is not central to our intended usage. Instead, there is another interface that has been developed related to the SOMLib system, though it does not form a core part of it: the SOMviewer.

The SOMviewer

This tool provides an intuitive interface to a 2-dimensional grid-like map. It allows the user to zoom in and out of the map, provides information like the number of documents mapped onto a specific unit and labels assigned to this unit by the LabelSOM method, and also incorporates a variety of different visualization options for the displayed SOM. Figures 2.9 and 2.10 show screenshots of the SOMviewer with different visualization options.

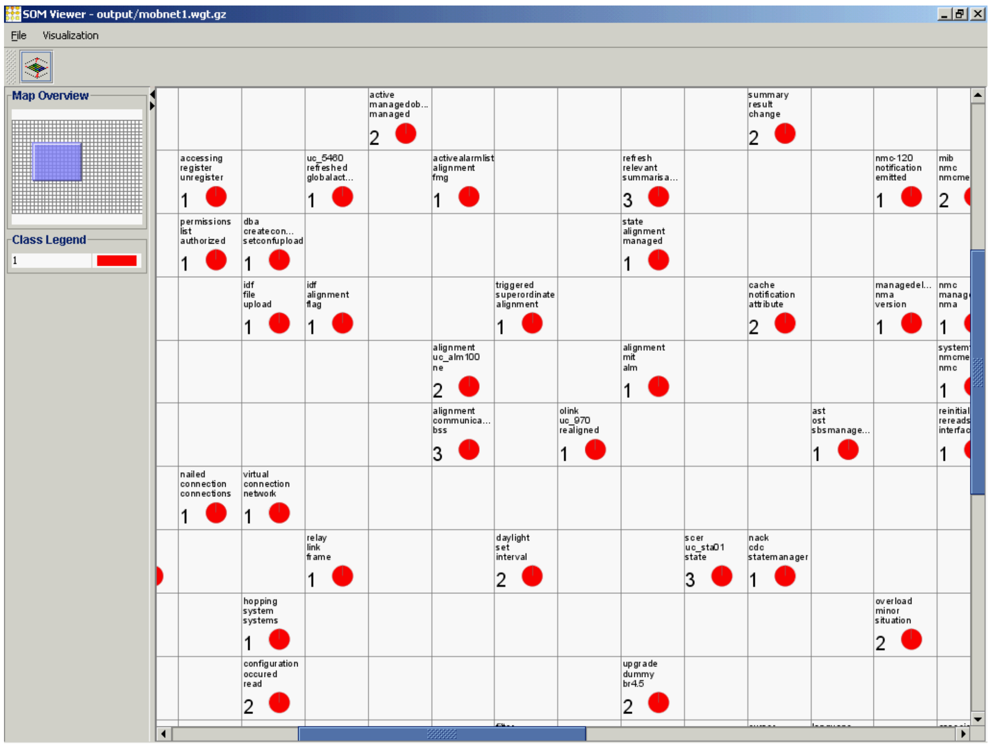

SOM Viewer - output/mobnet1.wgt.gz
File
Visualization
Map Overview
Class Legend
1
active managedob... managed 2
summary result change 2
accessing register unregister 1
uc_5460 refreshed globalact... 1
activealarmlist alignment fmg 1
refresh relevant summarisa... 3
nmc-120 notification emitted 1
mib nmc nmcme 2
permissions list authorized 1
dba createcon... setconfupload 1
state alignment managed 1
idf file upload 1
idf alignment flag 1
triggered superordinate alignment 1
cache notification attribute 2
managedel... nma version 1
nmc manag nma 1
alignment uc_alm100 ne 2
alignment mit alm 1
system nmcme nmc 1
alignment communica... bss 3
olink uc_970 realigned 1
ast ost sbsmanage... 1
reinitial rereads interfac 1
nailed connection connections 1
virtual connection network 1
relay link frame 1
daylight set interval 2
scer uc_sta01 state 3
nack cdc statemanager 1
hopping system systems 1
overload minor situation 2
configuration occured read 2
upgrade dummy br4.5 2

Figure 2.9: SOMViewer interface showing a collection of use cases.

1. The **U-Matrix** described by [Kra92, US89] uses the mean distance between weight vectors of neighboring units to visualize cluster structures with gray scales or color scales. Light shading indicates small, dark shades indicate large distances between vectors. Thus, a "landscape" of vectors is constructed that allows a human viewer to grasp clusterings more easily.

2. The spreading of data on the map is visualized using data histograms or **hit histograms** that show mapped input data on the map. In Figure 2.9, the circles represent mapped input units, with the number aside a circle giving the number of input vectors mapped to this unit.

3. An extension to hit histograms are **smoothed data histograms** [PRM02], where each data point is assigned to more than one unit in a fuzzy way that leads to a smoothed transition between clusters. This is depicted in Figure 2.10.

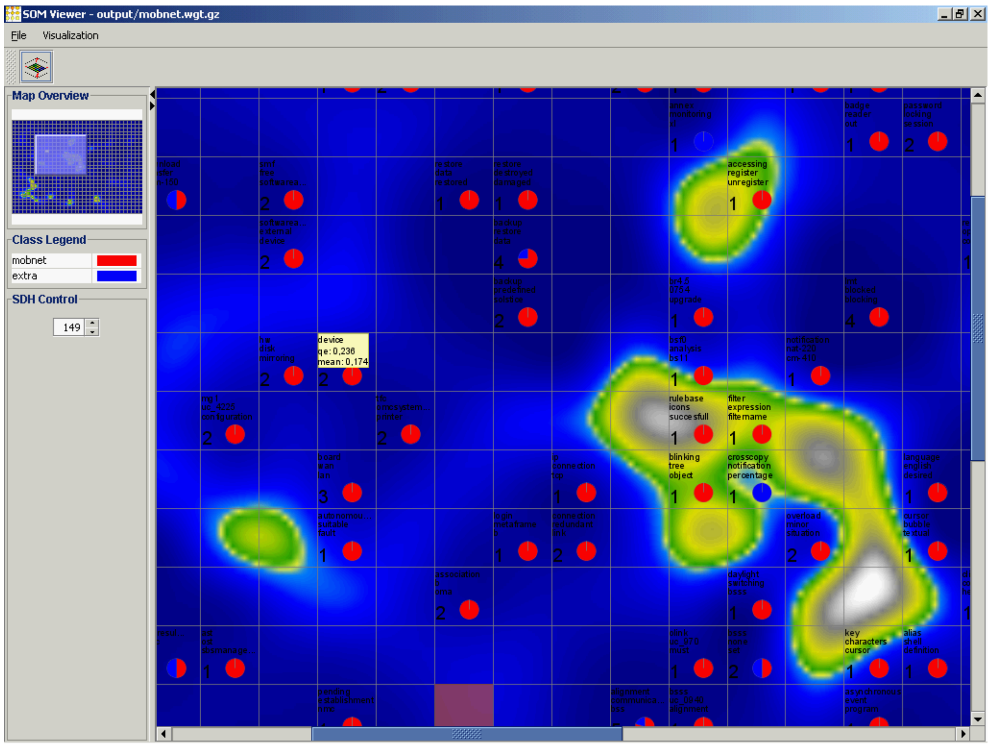


Figure 2.10: SOMViewer interface showing smoothed data histograms of a use case document collection.

2.2.4 Summary

In this section we introduced the SOMLib digital library system, its methods, paradigms, components and the corresponding workflow. We introduced the concept of self-organizing maps and varations thereof that improve both the flexibility in the representation as well as topology preservation. We described efforts aimed at using the self-organizing map for organizing document collections according to their content, and described the steps necessary to achieve this, in particular the workflow and components of the SOMLib toolsuite.

We will now take a look at the type of documents we are going to experiment with, namely use case descriptions, and further outline how clustering these documents might be be of actual value in industrial practice.

2.3 Use cases

The concept of use cases emerged in the early nineties and is discussed extensively in several textbooks [JCJv93, Coc00, OP04]. Cockburn describes this concept as follows:

> A use case captures a contract between the stakeholders of a system about its behaviour. The use case describes the system's behaviour under various conditions as the system responds to a request from one of the stakeholders, called the *primary actor*. ([Coc00],p. 1)

Similarly, Övergaard et. al. define use cases as

> ...defining how the modeled system is to be used by its surroundings. A *use case* models one usage of the system; that is, it describes what sequences of actions will be performed by the system as a response to events occuring outside the system caused by the users. ([OP04], p. 35)

Thus, a use case is a textual description of the behaviour of a system in reponse to outside events caused by the system users. Use cases have been applied successfully in a vast variety of fields, ranging from embedded systems [NMB02] and client-server supply-chain simulations [CBC$^+$99] to B2B e-commerce applications[SJP02]. They are not only successfully being applied for systems analysis, but also in areas like business process reengineering [JEJ94, Coc00].

A use case may be written in different forms and may contain several sections of text; which ones are actually contained varies greatly between projects and depends on a number of parameters like

- the style of writing,
- the intended audience,
- organization-wide policies, and
- the requirements of the project.

Some of the most common sections that may be present in a use case description are

- The **name** of the use case.
- The **scope** defines what is considered black box, i. e. the system under design.
- **Level** should be one of summary, user-goal, or subfunction.

   1. A **summary level** use case describes a (probably longer-term) overall goal of a user, like *Sell an item* in an online auctioning system.
   2. A use case at the **user-goal** level describes a necessary step for achieving a summary level goal, e. g. *Register user.*
   3. A **subfunction** use case in turn describes a lower-level goal that does not yield any direct benefit to the primary actor; it is necessary to complete a higher-level step. A common example is *Logon user.*

- The **Primary Actor** is the stakeholder requesting a service from the system in order to achieve a goal. Usually, the primary actor is the one triggering the use case.
- **Intent** captures the goal of the actor in a short statement.
- **Context of Use** may consist of a statement of the goal and possibly the circumstances under which this use case occurs.
- **Secondary or Supporting Actors** are other systems that are supposed to deliver subgoals for the system under design.
- **Preconditions** state the conditions that have to be met before the use case can execute. The most common example is *User is logged on.*
- **Minimal Guarantees** state what the system guarantees to be achieved even in case of a failure during the flow of events. Some common minimal guarantee would be *The system logs the progress of the use case execution.*
- **Success Guarantees** state what is achieved if the use case executed successfully. For the *Logon* use case, this might be *User is successfully authenticated and session initialized.*
- **Trigger** describes the event causing the use case to start. For a use case *Handle emergency call* in an emergency operation call center, this might be *Someone calls 911.*
- **Includes** states which use cases are included, or used, within the flow of this use case. E. g., a use case *Edit Something* may include a use case *Search for Something.*

- The section **Flow of events**, also named **Main Success Scenario** or simply **Description**, contains the main eventflow, the description of a series of steps that comprise a successful execution of the use case. Errors and alternative flows are usually handled separately, in the next section.

- **Extensions** defines alternative flows of events, e. g. when conditions cannot be validated successfully. If, for instance, in the above mentioned use case *Logon user* some steps of the main success scenario read *4. The system verifies the provided credentials... 5. The sytem grants access*, a possible extension would be *4a. The supplied credentials are not correct: The user is notified and may enter the credentials again.*

- **Management information** like priority, due date, history of changes, responsible author, frequency of occurence, etc., is sometimes attached to the use case description.

Cockburn suggests (among others) two different styles of writing a use case:

1. The **casual use case** form is often used in agile environments or during early stages of a software project. It contains only a few sections, e. g.
   (a) Primary Actor,
   (b) Scope,
   (c) Level, and
   (d) the description of the eventflow.

2. A **fully dressed use case**, on the contrary, features a much more thoroughly structured way of description; it may consist of a lot more sections like those mentioned in the listing above, and will usually use numbered steps to describe the flow of events and the extension list.

Furthermore, he suggests a simple way of employing some of the benefits of use case writing while at the same time investing minimum time, effort and standardization: Writing *use case briefs* that contain as little as the sections described in the casual case above or even less, with the description being just a few lines. These use case briefs can be of benefit especially in agile environments where communication between team members is very tight.

Of course, in reality one finds use case styles everywhere inbetween the range of variations described above. For examples regarding the sections that are likely to be found in real-world use case descriptions of industrial practice and how these differ in aspects like size, etc., please refer to Section 4.1.

2.4 Application areas of use case clustering

In 2004, the author was involved in writing the paper which initiated the research leading to this thesis. Auer et. al. [ABRB05] investigated the possibilities of using implicit analogies like textual similarities between use cases to facilitate the analogy-based cost estimation of software projects. We used the SOMLib digital library system to cluster use cases manually extracted from larger requirement documents that we obtained from industrial partners.

We argued that it should be possible for project managers to benefit from the advantages of a digital library system like SOMLib using self-organizing maps, namely

- the automatic unsupervised organization of documents according to their content, without the need for explicit ordering and intense human involvement,
- the topology-preserving ordering of documents on an intuitively graspable 2-dimensional map, and
- the possibilities for exploring yet unknown document collections and projects and thus gaining a fast overview of the inherent structure of projects and their functionality's cluster structures.

This implicit analogy can complement, or maybe under some circumstances even substitute, costly collected project metrics in aiding the decision processes in software engineering.

While the idea of clustering use cases seems promising, it is not clear if it can really help substantially in estimating software project effort. This is mainly because a similarity in content does not have to be correlated by any means to cost structures of implementing this functionality, and knowing functional clusters in itself does not mean knowing the effort needed to support these features.

Instead, there seem to be other areas that would benefit more from the content-based organization of use cases than cost estimation, in particular, *software reuse* and *knowledge management.*

2.4.1 Software Reuse

The concept of formal software reuse was first introduced by McIlroy in [McI69]. Prieto-Diaz [PD93] refers to [Fre83] and defines software reuse as

> the use of existing software components to construct new systems. Reuse applies not only to source-code fragments, but to all the

> intermediate work products generated during software development, including requirements documents, systems specifications, design structures, and any information the developer needs to create software.

Mili et. al. [MMYA01] take on a similar perspective when they define software reuse as

> the process whereby an organization defines a set of systematic operating procedures to specify, produce, classify, retrieve, and adapt software artifacts for the purpose of using them in its development activities.

Software Reuse has been an active research area for several decades and has been successfully adopted by a large number of organizations [MET02].

A central aspect in software reuse is the concept of software repositories [GL00], also called software component libraries or reusable software libraries, for storing, searching, and retrieving reusable software assets.

Examples for public software libraries range from large libraries like Sourceforge [Ope06] hosting open source software projects to collections of design patterns like the GoF design patterns[Tal06]. These repositories or libraries can also be seen as kinds of digital libraries as we described them in Section 2.1. In practice, the critical aspect with these reusable software libraries usually lies in the classification and retrieval process, in the process of *finding reuse candidates.*

A lot of research effort has been put into methods for the classification and retrieval of reuse candidates, like faceted classification schemes [PD91, PY93] that are now widely used in software libraries, also e. g. by Sourceforge, together with retrieval methods like full-text indexing and keyword search.

Most of these mehods rely on supervised classification schemes. However, there have also been topological approaches, and several researchers used self-organizing maps for organizing software libraries. [TS04]

- Merkl. et. al. [MTK94] used a self-organizing map to organize a small set of 36 MS-DOS commands according to their documentation. The feature space consisted of only 39 dimensions, the terms were extracted using a binary indexing scheme.

- Ye and Lo [YL01] clustered a library of UNIX commands, incorporating a thesaurus and a public synonym dictionary. They implemented a prototype and compared its performance to another public retrieval system.

- Tangsripairoj et. al. [TS05] use the more flexible growing hierarchical self-organizing map because, as they state,

  > the use of the traditional SOM . . . may not be practical when the number of software components stored in a software repository is large.

  The input they are using are C/C++ source code files, i. e. they do not rely on natural language descriptions of the software, but instead directly use the source code.

- Brittle et. al. [BB03a, BB03b] apply the self-organizing map to large software collections, building a distributed system to support interactive exploration of software collections; however, they do not present results.

The results of these studies suggest that using textual descriptions of software is a suitable approach to organize a software repository. Yet, detailed experiments involving datasets from industrial practice are needed to further investigate the applicability to real-world situations.

2.4.2 Knowledge Management and the Learning Software Organization

Another area which we deem suitable for employing the use-case clustering is knowledge management (KM) or, more specifically, the concept of the *Learning Software Organization (LSO).*

The Learning Software Organization as described by Ruhe et. al. is

> a continous endeavor of actively identifying (discovering), evaluating, securing (documenting), disseminating, and systematically deploying knowledge throughout the software develeopment organization. ([RB00b], p.3)

It extends an approach that has gained tremendous interest in research and practice, the concept of the so-called **experience factory** promoted by [BCM$^+$92, MPP$^+$94, VZM$^+$97]. The experience factory (EF) concepts enhances organizational learning in the domain of software engineering by providing an organizational infrastructure to support knowledge management, thus also forming a foundational framework for tackling the technological issues of KM.

> Objects of learning are all kinds of models, knowledge and lessons learned related to the different processes, products, tools, techniques, and methods applied during the different stages of the software development process. ([RB00b], p.6)

Two trends that have been pointed out by several researchers in this field are of particular interest here: Agility and tacit knowledge.

Agility

Doran [Dor04] reports on experience with knowledge management techniques in an agile environment. Because of the organizational burden and the sometimes prohibitive technological requirements of a full-fledged experience factory including organizational knowledge management, an experience base, etc., these heavy-weight approaches are often not viable for smaller organizations. In their keynote to the 2004 conference on Learning Software Organizsations [HM04], Holz and Melnik emphasized that

> . . . there is an increasing trend towards knowledge management approaches that are lightweight, i. e., do not introduce a considerable additional burden on developers and end users, while at the same time ensuring that the hoped for experience factories do not become “experience cemeteries” which no employee uses.

Because of their high degree of automation and their ability to operate directly on documents that are being created anyway during the course of software projects, SOM-based digital libraries may be a convenient method with a low-entry barrier to allow for organizational learning, as long as tool support for organizing documents into collections is sufficiently effective.

Tacit knowledge

Tacit or hidden knowledge is the part of knowledge that is not codified in an explicit form. As Horvath explains,

> lots of valuable knowledge “falls through the cracks” within business organizations, never finding its way into databases, process diagrams, or corporate libraries. As a consequence, much of what the firm “knows” remains unknown or inaccessible to those who need it. Such knowledge is present within the organization, but it remains hidden, unspoken, tacit. In business organizations, this hidden or tacit knowledge takes one of two forms: 1) knowledge embodied in people and social networks, 2) knowledge embedded in the processes and products that people create. ([Hor], p. 3)

Several authors point out that tacit knowledge forms the most valuable experience in an organization.

- Ruhe et. al.[RB00b] point out that one of the main challenges facing the LSO community is

  > to extend the knowledge management approach of the EF to also handle the tacit knowledge available within an organization. ([RB00b], p.4)

- Horvath points out that

  > some of the most valuable knowledge within a firm is essentially hidden or tacit – residing not in documents or databases but in the experience and skill of human beings. ([Hor], p. 2; cf. [Hor00])

- Similarly, Johansson and his colleagues [JHC99] claim that

  > the most valuable experience is tacit. . .

  but, opposed to others laying major focus on experience databases, they further add that this valuable tacit knowledge is usually not captured in a database, but stays within the humans:

  > the most valuable experience is tacit and stored on the individual level. ([JHC99],p.171)

  They focus on pointing the user looking for information to the right source, which is usually a human being.

  > Our approach . . . focuses on mediating referrals to sources holding the correct expertise – usually human sources.

Conclusion

Using automatically constructed self-organizing libraries of software artifacts, e. g. use case descriptions, instead of costly built databases, might have two main advantages:

1. A light-weight approach could be beneficial especially to smaller organizations that do not have the resources to start a demanding knowledge management initiative. It could enable these organizations to start an initiative which they probably would not undertake otherwise.

2. It would be very easy and simple to integrate approaches like the experience engine proposed by Johansson et. al.[JHC99] with the organized document collections. Because this approaches could initially rely solely on the documents already present in an organization, even small organizations following agile development patterns without maintaining extensive knowledge databases und documentation could benefit from this agile knowledge management approach.

Though the ideas might be promising, further investigation is needed, especially on the practical applicability of approaches based on these ideas. Improving tool support is an important step in this direction.

2.5 Summary

In this chapter we introduced the concept of digital libraries and described the SOMLib digital library system, which relies on self-organizing maps to achieve a content-based organization of document collections without the need for supervision.

We then discussed the concept of use case descriptions and outlined two areas that might benefit from clustering use case documents using the SOMLib system, namely software reuse and knowledge management.

The next chapter will outline the current deficiencies in tool support regarding the workflow of pre-processing, organizing and viewing documents, and propose tool enhancements to improve the situation. The SERUM toolsuite and it's development is then described in detail.

Chapter 3

The SERUM tool suite

This chapter presents the tool SERUM – SElf oRganizing Use case Maps. We start by explaining the need to develop such a tool and discussing the requirements for SERUM. After a description of the supporting tools and frameworks that were used, we present the overall architecture, the domain model and the building blocks of SERUM. These components are then explained in detail. The chapter is concluded by a summary.

3.1 Current deficiencies in tool support

Digital libraries are an important part of the information age we are living in. In the last chapter we introduced the concept of digital libraries and presented an overview of the SOMLib digital library system.

While this system is offering a number of advantages and possibilites for content-based organization, there are still several problems hindering both wider accepted usage in practice and an easier exploration of, and experimentation with, the possibilities the system provides us with.

In particular, the following obstacles stand in the way of a user or an organization that wants to reap the benefits of employing self-organizing maps to organize document collections by using the SOMLib digital library:

1. The **extraction process is tedious**. To apply the SOM algorithm, feature extraction has to be done in the form of text indexing that transforms texts into a numerical representation suitable for the SOM. This process usually works on single files. But relevant document passages in real life are contained in larger documents, which have to be split up first in order to allow text indexing. So far, this has to be done in a manual preprocessing step that can take quite a long time and may be prohibiting in terms of effort.

Furthermore, often texts are being clustered that are themselves structured in various segments. For example, a scientific paper usually consists of

- A list of authors,
- the title,
- an abstract,
- a list of keywords,
- several sections of text, and
- a listing of references.

Similarly, a use case description might contain sections like

- A name,
- actors,
- event flow description,
- pre- and post-conditions,
- etc. (see Section 2.3)

With the tools currently used, all these sections are used together to organize documents according to their similarity as provided by the word frequency distributions. But if we take a closer look at the sections of these documents, we can easily see that they are not equally important, that each section contains text of very distinct characterstics, and corresponding sections across different text documents are often very similar in structure and content. By using different sections of texts or combinations of sections – e. g., the preconditions, successconditions and failure conditions of use case descriptions – and possibly comparing clusterings according to different combinations of sections, we might get a much better insight into specific aspects of these document collections.

To explore this, however, we need a tool that allows us to flexibly configure these combinations without the need for manual and repeated partitioning of the texts.

Another aspect of the extraction process is the performance of indexing documents. While it is not taking hours to index a few hundred use cases, it still takes several minutes, with larger collections taking prohibitively long for a comfortable workflow.

2. Tool support **automating the workflow** of feature extraction, indexing, and training is yet quite poor. The user has to manually handle the input and output files of the various stages and feed the input files of preceding steps to following tools.

3. Tool support **linking the user to the content of the original documents** is not comfortable. While both the libViewer and the SOMviewer include a link to the original document, these merely open the original documents in a dedicated viewer. Especially in the common situation where the text that is mapped onto a unit on the map has been extracted from a larger document containing a number of texts, this approach is not sufficient. For example, when 300 use cases contained in one `pdf` file are being organized in a map according to the content of their pre- and postconditions, simply putting a reference to the `pdf` file clearly is not enough.

   What is needed is a flexible and comfortable way to display the original texts, respectively parts of them like the sections actually used for clustering, directly inside the map browsing application. This includes both a quick preview while browsing the map, as well as the option to view the complete text directly in the map browser.

This thesis strives to overcome the deficiencies explained above. It focuses on tool enhancement for the SOMLib digital library system to facilitate an automated workflow ranging from the extraction and indexing to the map interaction stages by addressing the issues mentioned above and provide further support. The tool to be developed is named "**SERUM – SElf oRganizing Use case Maps**".

3.2 The goals of SERUM

Goal is the **further development, enhancement and integration** of the existing components of the SOMLib system to a tool that allows the user in a comfortable way

- to **import** in a way **as automated as possible collections of segmented documents containing short texts** (in particular requirements specifications containing use case descriptions),
- to manage these document collections and **organize them according to configurable criteria in SOMs**. The resulting maps should be saved, and an extended SOMviewer shall present them in a convenient and helpful way and thus **support the analysis process**.

For use case documents, e.g., the user defines templates with specified parts and keywords. The system imports use cases according to these patterns, trains SOMs according to the user's needs and presents them to the user for browsing and exploration, which includes viewing the text of the original use cases.

Of course, depending on the types of texts to be organized, users of SERUM will have different backgrounds and requirements. We concentrate here on the needs of two groups of people:

1. People working in a **team on a software project** who capture the requirements of their projects with use cases want to extract the use case descriptions from their documents and analyze them with self-organizing maps. These are in particular project managers, software architects, reuse specialists, developers, etc.

2. **Researchers** working with different types of document collections, possibly also use case documents, want to extract the short texts contained in their documents and analyze them with self-organizing maps. In particular, they might be interested in

   - the repeated and possibly automated execution of tasks like feature space pruning,
   - the tuning of parameter settings like index limits or parameters controlling the growth process of a self-organizing map, and
   - an efficient way to configure pattern matching criteria for the segmentation of documents.

While starting our requirements analysis with scenarios involving Use Case documents, we try to keep the resulting usage scenarios as general as feasible to facilitate different tool applications. Furthermore, we implement a generic solution, thus making it possible to use the tool for entirely different collections of segmented texts.

Specifically, we want to improve and integrate existing components, complement them with additional modules and enhance the execution of the following steps of the workflow:

1. **Feature extraction from documents**

   SERUM should allow a comfortable way of feeding documents into a collection. In particular, it should work on the documents provided on an as-is basis without further pre-processing needs like converting the documents to flat text files, manually extracting texts contained in larger documents, etc.

This involves the following steps:

(a) **Extract text artifacts and sections of texts out of larger documents.** – This means allowing the user to import a large document containing lots of pieces of text, e. g., a requirements document available in `doc` or `pdf` format that contains a number of use case descriptions.

These use cases and their constituting sections shall be extracted automatically and saved. Due to the nature of natural language text written by human beings, which does not always follow the exactly same pattern, this extraction process has to be supervised by the user, allowing her or him to view the results of extraction and correct them efficiently.

(b) **Create a number of maps with different criteria, combination of input sections and configuration for the same document collection.** – The user shall be able to create a number of distinct maps, e. g. one map clustering use cases according to their similarities in the sections describing non-functional requirements, another one organizing them according to the content of the eventflow, etc., while keeping an overview of the various maps and the sections included in each map.

(c) **Extract feature vectors from documents and configurable sections of documents.** – This feature allows the user to intuitively select which sections of texts to include in the organization of documents in a map, and apply the feature extraction on these sections.

2. **Efficient and effective feature space reduction.** – This step automates the pre-processing necessary to reduce the feature space. It should be possible to

   (a) **perform fully automated feature space reduction** based on heuristics or preset defaults to be incorporated into the tool, as well as

   (b) **support sophisticated fine-tuning** of the feature space reduction by allowing the user to adjust settings like stop word limits and the usage of stop word lists, while at the same time previewing the effects of these adjustments for rapid feedback.

3. **Comfortable integration of map training.** – The training process should be initiated directly in the tool, allowing the user to choose

the desired network model to use, set parameters like the number of iterations or the labelling to be applied, etc.

During the training process, feedback on the progress of the training process shall be provided to the user.

4. **Display the input documents in the map.** – SERUM should provide the user with quick information on the content of cells in a map and be able to display the contents of texts mapped onto a specific unit that the user has selected on the map.

5. **Unify all steps within one tool.** – All of these steps should be integrated into one application to enhance the workflow. Only the map browsing is being kept separate, both due to technical limitations and due to the reasoning that often the ones who explore maps will not necessarily be the same users as those who import and organize documents.

6. **Spare the user from manually handling input and output files.** – The user should be able to handle the whole process in a comfortable way without editing a file other than the pattern configuration of document types.

7. **Provide batch processing options.** – To facilitate the automated processing of large amounts of data according to predefined configuration criteria for document and feature extraction, feature space reduction and map training, batch processing should be possible. This option is aimed at advanced users.

The next sections will discuss the requirements for the tool in detail, also from a user perspective by describing them with use cases, and present the components of the tool and their collaboration.

In chapter 4, we then use the tool to cluster use case descriptions in a SOM. We describe the feature extraction and feature space reduction steps and the workflow that is employed as well as the resulting maps. We further point out possible applications to software reuse and knowledge management.

3.3 Requirements

Similar to the workflow described in Section 2.2.3, the four main steps in using SERUM as depicted in Figure 3.1 are

1. **Import documents into collections**, which includes the segmentation into the contained texts and sections;
2. **index collections**,
3. **train maps**, and
4. **interact with the trained maps**.

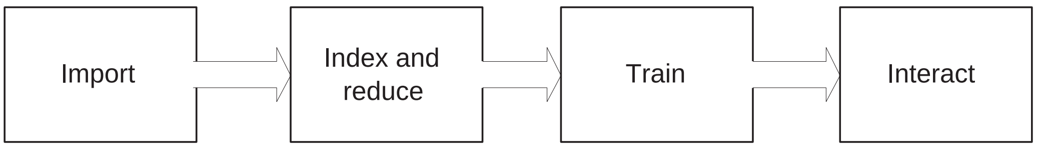


Figure 3.1: Four main steps of SERUM.

Based on these steps, we can derive the main use cases, unfolding from high-level goals to lower-level functions. We will provide use case descriptions for the main tasks requested by the user, and proceed with presenting the components that support these tasks in the corresponding order.

3.3.1 Use Case Descriptions

The following section lists the use cases describing the functionality covered by SERUM. We will not produce fully elaborated use case descriptions with extensions, includes, etc. in this work; these use case descriptions are meant to give the reader an overall view of the functionality provided by SERUM and thus contain mainly the so-called *main success scenarios* and the most central sections of a use case description. We thus follow the **casual use case** approach as desecribed by Cockburn [Coc00] and explained in Section 2.3. While being rather informal, this approach is the most appropriate for the environment in which SERUM is being developed, and allows for an easy overview of the functionality supported by SERUM.

Underlined phrases in the use case descriptions significate that a lower-level use case is being included.

The use cases we will describe are:

UC 1. **Use SERUM**
This high-level summary use case is described as an entry point to the included use cases as suggested by Cockburn[Coc00]. Its casual description is provided in Table 3.1.

UC 2. **Import a collection of documents**
This use case, described in Table 3.2, constitutes the first step of the workflow, where texts contained in larger documents are imported into SERUM according to specified patterns.

UC 3. **Create a map for a document collection (normal)**
The standard procedure for creating a map is described in Table 3.3. For convenience, and also for inexperienced users, a more automated way of map creation is modelled in

UC 4. **Create a map for a document collection (simple)**
This simplified method, described in Table 3.4, relieves the user of the details of feature space reduction.

UC 5. **Browse a map**
This allows the user to explore a trained map using the SOMViewer. It is described in Table 3.5.

UC 6. **Organize a collection of documents in a map (fully automatic)**
For the convenience of (primarily novice) users, this "full-service" feature, described in Table 3.6, does everything from importing documents to training the map by itself, providing the user a nearly instant access to the resulting map.

Details of the functionality and workflow that SERUM provides will be discussed in the next chapter.

3.3.2 Non-functional requirements

A number of goals describing non-functional requirements and constraints had to be considered. The most relevant are

- usability,
- performance,
- platform independence, and
- easy distribution.

UseCase	Use SERUM
Level	Summary
Precondition	Template for use case extraction has been defined
1.	User starts SERUM.
2.	User creates a new project for a text type, i.e. the Use Cases as predefined in the company template.
4.	User imports a collection of documents containing use case descriptions matching the company use case pattern into the project.
5.	User creates a subproject for this collection with all segments included and trains a map.
6.	User browses the map.
7.	User exits SERUM.

Table 3.1: UC1: Use Serum

Usability

One of the main goals in developing SERUM is to ease the workflow and relieve the user of the burden of tedious, repetitive tasks, automating as many of them as possible. Thus, an *intuitive user interface* that is easy to understand and use, together with an *accessible workflow* that does not require long training, is essential.

Performance

The steps of extraction, indexing, feature space pruning, and training should take no longer than a few minutes for an average collection consisting of several dozens to a few hundreds of use cases, as long as the pattern for extraction has already been defined. In particular, it should be feasible for the user to wait for the steps to complete, instead of working with the tool asynchronously. Otherwise, the workflow would be interrupted, which is considered uncomfortable by most users. This also calls for feedback on the progress of work to the user.

The process of defining the patterns for text extraction out of documents is not as critical, because it is typically performed more rarely and by more experienced users. A maximum of a few hours is considered to be sufficient; moreover, extracting text documents from ill-formatted documents within a few minutes is not feasible.

UseCase	Import a collection of documents
Level	User
Precondition	Template for use case extraction has been defined. The user has created or opened a project.
1.	User selects to import a collection of documents.
2.	User selects • the document type, • the category (e.g., a project) that should be assigned, and • the file or directory to import.
3.	SERUM presents information about the amount of texts that were found.
4.	User verifies amount and initiates segmentation.
5.	SERUM segments documents and extracts the texts while providing information on its progress by displaying • the number of texts correctly parsed, • the number of texts that do not confirm to the pattern, and • for each text that cannot be parsed: the content.
6.	The user verifies the segmentation by browsing the list of extracted texts and segments, and initiates the writing of the segmented texts.
7.	SERUM writes the texts and segments to the filesystem.

Table 3.2: UC2: Import a collection of documents

UseCase	Create a map for a document collection (normal)
Level	User
Precondition	The user has opened a project which already contains some documents.
1.	User selects to create a subproject and enters a name.
2.	SERUM creates a subproject and activates it; it displays a list of sections available for indexing.
3.	User selects which segments of the text type shall be used for content-based organization and initiates indexing.
4.	SERUM indexes texts constituted by aggregating the selected sections.
5.	The user reduces the feature space by setting word limits and viewing the effects of his adjustments.
6.	SERUM saves the vector space model as edited by the user.
7.	The user sets the parameters for map training and initiates the training process.
8.	SERUM trains a self-organizing map while providing feedback to the user.

Table 3.3: UC3: Create a map for a document collection (normal)

UseCase	Create a map for a document collection (simple)
Level	User
Precondition	The user has opened a project which already contains some documents.
1.	User selects to create a subproject and enters a name.
2.	SERUM creates a subproject and activates it; it displays a list of sections available for indexing.
3.	User selects which segments of the text type shall be used for content-based organization and initiates automatic indexing.
4.	SERUM indexes texts constituted by aggregating the selected sections, and automatically reduces the feature space. It provides information on the results and suggests parameters for the training of a self-organizing map.
7.	The user may adjust the parameters for map training and initiates the training process.
8.	SERUM trains a map while providing feedback to the user.

Table 3.4: UC4: Create a map for a document collection (simple)

UseCase	Browse a map
Level	User
Precondition	The user has opened a project which already has a map associated.
1.	User selects to browse a map.
2.	SERUM opens the SOMViewer with the selected map.
3.	User interacts with the map and <u>selects visualizations</u>. While browsing, the system shows information on currently selected cells in an appropriate degree of detail.
4.	At any time, the user may <u>view a document</u> mapped onto a cell.
5.	User exits SOMViewer.

Table 3.5: UC5: Browse a map

UseCase	Full-service SERUM
Level	Summary
Precondition	Template for use case extraction has been defined, SERUM is running, a project is active
1.	User selects a document type and a file to import and triggers full-service action.
2.	SERUM imports the document and prompts the user for a name for the map to create.
3.	User provides a map name.
4.	SERUM creates a subproject with this name for the active project with all segments included, indexes it and reduces the feature space, trains a map, and finally opens the map in the viewer application.
6.	User browses the map.

Table 3.6: UC6: Full-service SERUM

Portability

The tool should be platform-independent respectively portable as far as possible to allow usage on different systems. The fact that the existing components of the SOMLib system are implemented on the Java platform greatly supports this goal.

Easy and fast distribution and installation

SERUM should be quickly distributable in a fast way to lower the entry barrier required to use the tool. Thus, the installation procedure, if required, should be short, and it should not be necessary to rely on additional components that have to be installed, like a relational database for the persistence layer.

Some of these requirements had to be traded off against each other in order to provide maximum value to the user. In particular, usability was considered to be more important than complete platform independence and fast installation.

3.4 Supporting frameworks and tools

A wide variety of frameworks and tools, most of them coming from the open-source community, supports the developer in the modern Java world. These frameworks relieve us from a lot of work, while being both freely available and easy to integrate in an application simply by including the `jar` file of the library. This section describes supporting components and frameworks that are used by SERUM:

1. **Logging Framework: Log4J** [Fou05]
   The open-source log4j framework is the most prominent logging framework in the Java world. It features a flexible configuration, high-speed logging and extremely flexible options of logging to text files, XML files, databases, and even sockets that may be attached directly to a log viewer. Specifically the **Chainsaw log viewer** [Fou06b] developed within the Log4j project is used to provide the users with detailed feedback on the extraction process.

2. **Object mapping: Jakarta Commons Digester** [Pro06a]
   The Digester component allows us to define a mapping for the XML file we are using to configure the patterns used for extracting texts out of documents. We use it to directly create a hierarchy of objects from the XML document. This is described in detail in Section 3.7.

3. **GUI toolkits: Standard Widget Toolkit (SWT) and JFace** [Fou06c, SBJ04]

   The Standard Widget Toolkit, implemented as part of the Eclipse Tools Project, is one of the leading GUI toolkits for Java. It works in a hybrid way, standing between completely platform-independent toolkits like AWT and Swing and proprietary toolkits designed for a specific platform.

   > SWT is an open source widget toolkit for Java designed to provide efficient, portable access to the user-interface facilities of the operating systems on which it is implemented. [Fou06c]

   SWT uses the JNI[1] features of Java to call native functions of the Operating System it is running on, while shielding users and developers

[1]`http://java.sun.com/j2se/1.4.2/docs/guide/jni/index.html`

from these dependencies. Because we consider usability to be more important than complete platform independence and fast installation, the SWT fits the requirements of SERUM best; it forms a trade-off exchanging platform independency for portability, getting standard look-and-feel and significantly improved performance in return. Though an SWT runtime does need to be installed on some platforms, this can be boundled easily with the distribution and even be deployed using Java Web Start[2] [Gun03]. Moreover, SWT supports all major operating systems.

JFace is an extension to SWT, similar as Swing is based on AWT. It allows an MVC-based programming style featuring sophisticated widgets like tree and table viewers.

The Jigloo GUI Builder [Clo06] was used to design the user interface.

4. **Parse document formats: Jakarta POI and pdfBox** [Pro06d, Pro06c]
For parsing documents in office formats like PDF and DOC, we use the possibilities of the Jakarta POI and pdfBox libraries that are able to extract the content of files in these formats.

5. **Indexing: Apache Lucene** [Fou06a]
The Lucene project develops an open-source framework for search engines. We use the indexing possibilities of Lucene for our feature extraction module.

6. **Detect language automatically - Java Text Categorizing Library**[Sol06]
When indexing texts, taking account the language of them can improve results for two reasons:

   - Word stemming algorithms can be used that are tailored for this language, and
   - language-specific stop words can be used.

   The JCTL library supports this by automatically detecting the language of texts by an *n-gram* algorithm [CT94].

7. **Supporting frameworks: Jakarta Commons**
Several libraries developed by subprojects of the Jakarta Commons project are used, partly because they are required by other Jakarta

[2]`http://java.sun.com/products/javawebstart`

frameworks we are using; e. g., the Digester relies on the *Beanutils* and *Collections* libraries. [Pro06b] provides detailed descriptions of these frameworks.

(a) **Logging** wraps common logging implementations including the Java Logging API and log4j, allowing to switch between these without having to change the source code.

(b) **Beanutils** wraps the reflection and introspection APIs of Java in a convenient way.

(c) **Collections** extends the possibilities of the collection classes included in Java.

(d) **Discovery** provides functionality for resource discovery.

(e) **Math** contains components complementing the Java language with often-used mathematical and statistical features.

3.5 Tool architecture

This section presents the components of SERUM and the way they collaborate to achieve the desired functionality.

3.5.1 Components

SERUM mainly consists of the following components, corresponding to the workflow as provided in Figure 3.1:

1. The **Extractor** retrieves texts and their constituting sections from documents according to configurable patterns by using regular expressions constructed from the user-defined XML configuration.

2. The **Indexer** uses functionality provided by Lucene to write index files of word frequencies. It also encapsulates the next module.

3. The **Feature space reduction** module is based on the **TeSeT – TErm SElection Tool** developed by Andreas Pesenhofer at the eCommerce Competence Center.[3] It supports the user in the feature reduction process, allowing both a fully automated reduction as well as manual fine-tuning of the reduction process.

[3]The ec3 homepage can be found at `http://www.ec3.at/`.

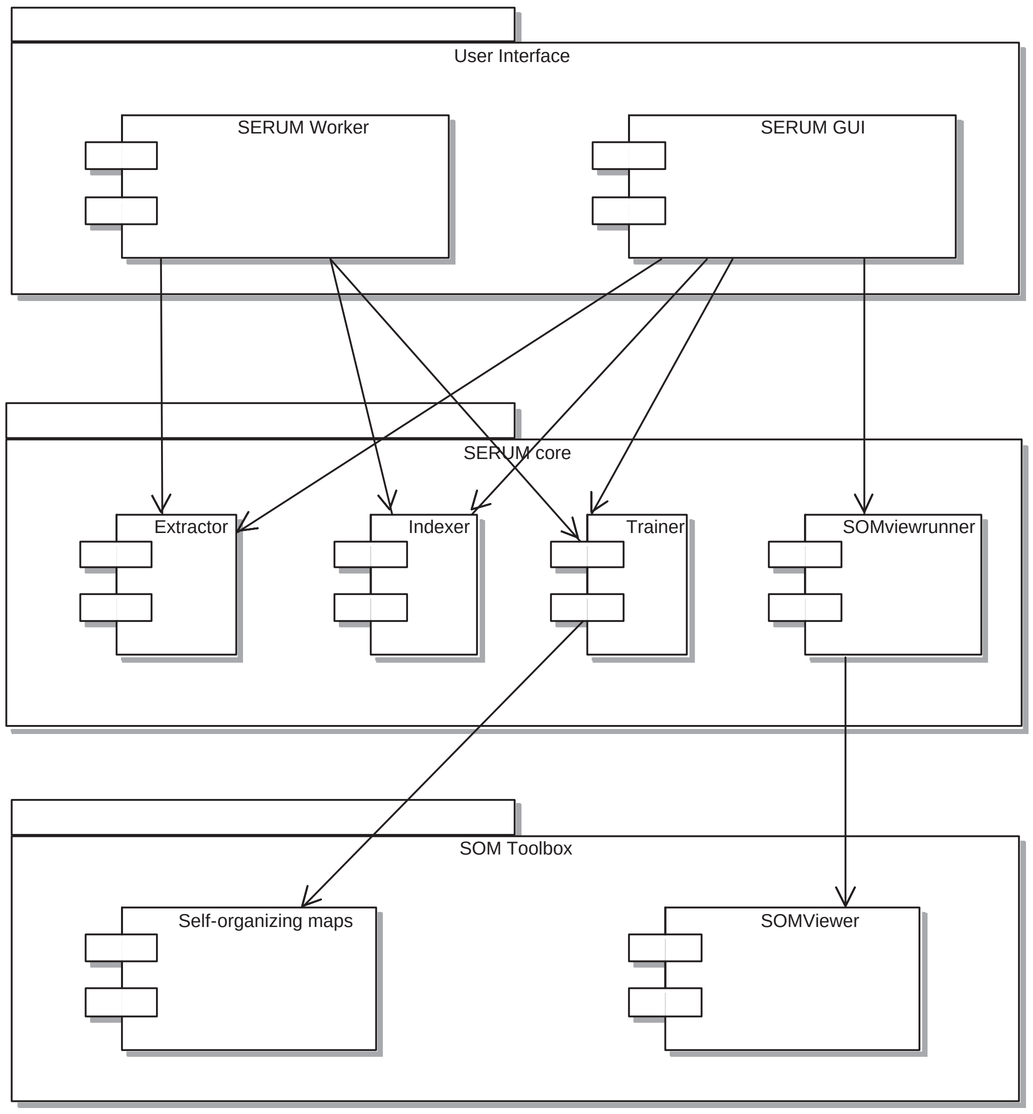


Figure 3.2: Overview of the components of the SERUM toolsuite.

4. The **Map trainer** controls the training processes of self-organizing maps as provided by the SOMLib library.

5. The map training process itself is executed by the existing **SOM Toolbox** that is being slightly adapted and extended for SERUM.

6. SERUM also contains an extended version of the **SOMViewer** customized to provide a comfortable way of displaying the documents mapped onto units on the map.

7. The complete workflow can be exercised both by a **graphical user interface** and a **batch-processing mode**.

Figure 3.2 provides an overview of the components and their relationships.

We will now present the main classes in our domain model and then discuss the components in the order of their appearance in the workflow.

3.6 The domain model

Figure 3.3 shows the main elements of our domainmodel. The elements in the lower right corner form the main concepts in the workingspace of SERUM, while the other classes are used for the processes of document extraction and import.

A **Project** is the container which holds the documents that we want to organize in different maps. These documents are imported as described in Section 3.7 and usually confer to the same doctype. A project maintains a list of subprojects and a list of all segment names occuring inside the project.

A **Subproject** represents the configuration of a self-organizing map to be trained with the set of documents in the project. The map can be organized not only according to the content of documents as a whole, but also according to the similarity of specific sections or segments, like the pre- and postconditions of use cases. Each subproject thus handles the configuration of segments to include in the indexing and training process.

A **Doc** represents a single document contained in a collection, e. g. a scientific paper or the description of a single use case. It consists of several segments, each having name and content. In the document pattern, this is represented by a prefix, followed by a body of text. E.g., a segment could be "*Preconditions: User is logged on*", where *Preconditions:* is the prefix and *User is logged on* is the content.

A Doc also belongs to a *category* that can be assigned during the import process and is used to discern documents when browsing the map. For a use case, this would normally be the name of the project.

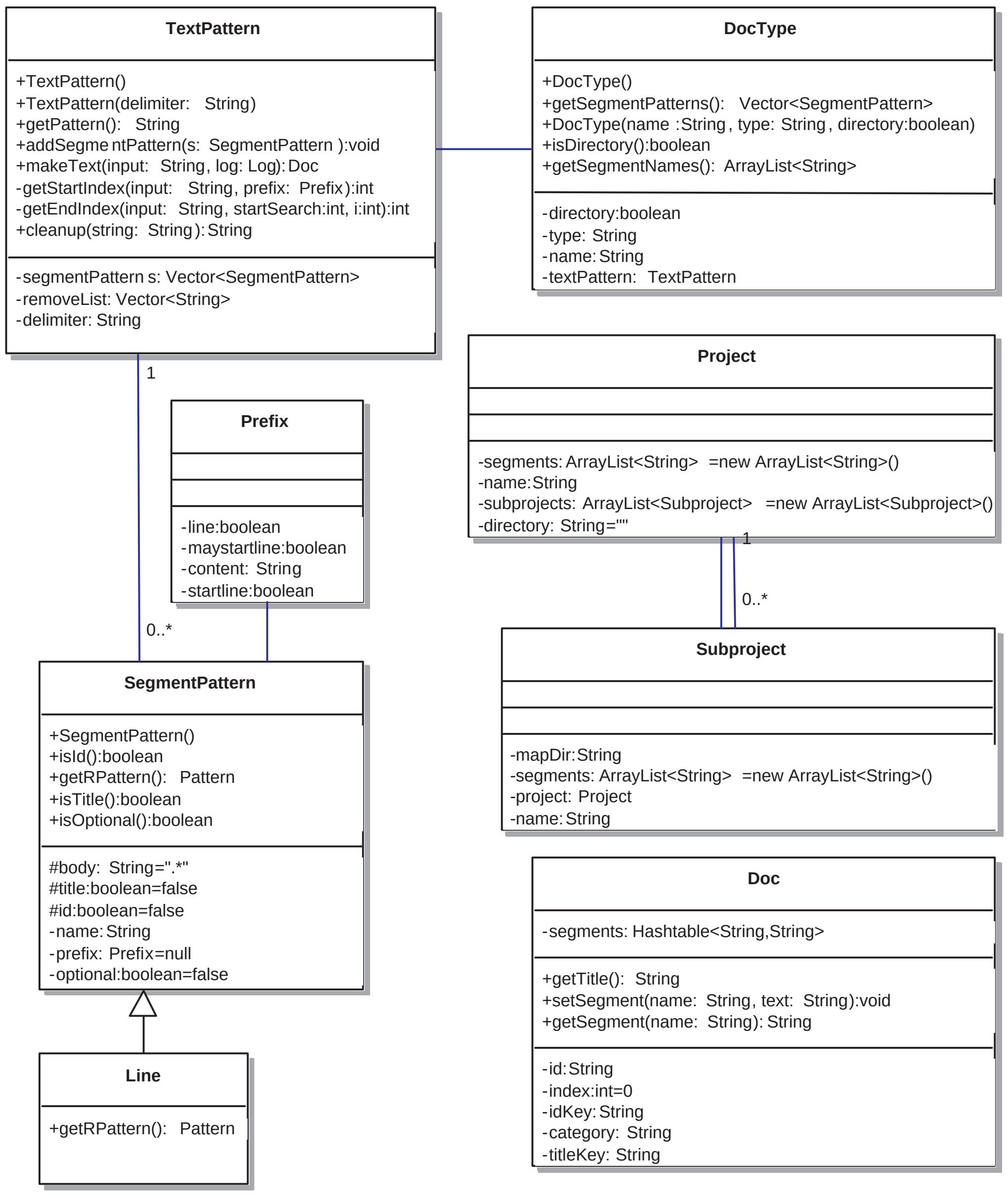

TextPattern
+TextPattern()
+TextPattern(delimiter: String)
+getPattern(): String
+addSegmentPattern(s: SegmentPattern):void
+makeText(input: String, log: Log): Doc
-getStartIndex(input: String, prefix: Prefix):int
-getEndIndex(input: String, startSearch:int, i:int):int
+cleanup(string: String): String
-segmentPatterns: Vector<SegmentPattern>
-removeList: Vector<String>
-delimiter: String
DocType
+DocType()
+getSegmentPatterns(): Vector<SegmentPattern>
+DocType(name :String, type: String, directory:boolean)
+isDirectory():boolean
+getSegmentNames(): ArrayList<String>
-directory:boolean
-type: String
-name: String
-textPattern: TextPattern
1
Prefix
-line:boolean
-maystartline:boolean
-content: String
-startline:boolean
0..*
Project
-segments: ArrayList<String> =new ArrayList<String>()
-name:String
-subprojects: ArrayList<Subproject> =new ArrayList<Subproject>()
-directory: String=""
1
0..*
Subproject
-mapDir:String
-segments: ArrayList<String> =new ArrayList<String>()
-project: Project
-name: String
SegmentPattern
+SegmentPattern()
+isId():boolean
+getRPattern(): Pattern
+isTitle():boolean
+isOptional():boolean
#body: String=".*"
#title:boolean=false
#id:boolean=false
-name: String
-prefix: Prefix=null
-optional:boolean=false
Doc
-segments: Hashtable<String,String>
+getTitle(): String
+setSegment(name: String, text: String):void
+getSegment(name: String): String
-id:String
-index:int=0
-idKey: String
-category: String
-titleKey: String
Line
+getRPattern(): Pattern

Figure 3.3: SERUM's domainmodel.

The remaining classes collaborate in extracting the Docs from files:

- **DocTypes** and **TextPatterns** are created from a user-defined XML pattern. They model the structure of a document containing texts that the user wants to import by defining
  - the delimiter separating single Docs, and
  - the structure and patterns inside these texts that define the contained segments.
- This structure is modelled by a **list of SegmentPatterns** contained in the `TextPattern`, each of which contains a prefix and a body. These are defined in the doctype XML file by a combination of regular expressions and attributes, as we will describe in the next section. A **Line** is a special kind of SegmentPattern modelling a segment that fits into one line.

3.7 Extracting text artifacts from documents

The indexing process, as supported by the existing code from the Term selection tool TeseT that we build upon, relies on accessing single text, `.pdf` or `.doc` files. However, in reality document passages like use cases are usually maintained in a requirements document containing dozens or hundreds of use case documents. Thus an important step has to be carried out before we are able to construct an index: the segmentation of larger documents into the smaller parts that interest us, or more specifically, the extraction of texts out of larger documents according to specified patterns.

The most common method to achieve tasks like this is to use regular expressions, which are readily supported to a sufficient extent by the native features of Java. They provide a flexible and powerful way of text pattern matching and extraction.

3.7.1 Pattern definition

For short texts or simple structures, directly using regular expressions might be sufficient, at least for experienced users. However, a lot of complexity is added by the irregularity that the human factor brings into natural language texts: Sections may be missing, section headings may be spelled or written differently within a document, or markup elements like colons and the like may be missing. Figure 3.4 gives an example of a regular expression necessary

```
(Goal in Context.*)?Use\s+Case\s+UC_.*(Covered\s+feature\s+
.*)?(()?Goal in Context.*)?Creator(s)?\s+(Responsible\s)?
.*History of\s+(Changes\s+)?.*Scope.*Preconditions.* Success
End\s+Condition\s+.*Failed End\s+Condition\s+.*Primary Actor.*
Secondary\s+Actor\(s\)\s.*Trigger.*\sSCENARIO SHEET\s+ (Use
Case\s+Scenario\s+)?Step\s+Action\s?.*(()? Extens(t)?ion to\s+
Scenario\s+Steps\s+Step(\s+Branching\s+Action)? \s{0,2}.*)?
(()?Variation(s)? to\s+Scenario\s+Steps\s+Step(\s+Branching\s+
Action)? \s{0,2}.*)?(()?Extens(t)?ion to\s+Scenario\s+Steps\s+
Step(\s+Branching\s+Action)?\s{0,2}.*)?(Superordinates .*)?
(Subordinates.*)?\s+QUALITY ISSUES\s+Priority:?\s?.*Time
Constrain(ts)?.* Frequency.*((Channels)|(connections))
to\s+(actors)? .* OPEN\s+ISSUES.*Due Date.*...any
other\s+(management \s+ (i|I)nformation...)?\s?.*
```

Figure 3.4: Regular expression for matching a complex requirements document containing several hundreds of use case descriptions.

to extract use cases from a rather complicated requirements document from industrial practice. We will discuss this document in detail in Chapter 4.

Forcing the user to define this pattern as a single complex regular expression string would not be very comfortable; besides, this approach would not allow the program to extract the single element bodies and assign them to named sections. Instead, building the document pattern hierarchically enables the desired extraction and makes the pattern definition much more flexible. Complexity is reduced to a large extent, as we will see in the following and in Chapter 4.

The user describes document patterns with short elements in an XML file, each one defining the pattern of a specific segment of text occuring in the document, all in their order of occurence. The segments contain a prefix and a body, the latter one defaults to match any string if it is not provided. By using XML attributes like `optional="true"` instead of regular expression constructs, overview and readability are improved. These XML attributes are later converted to regular expressions during the extraction process.

A passage of the XML file leading to the regular expression given in Figure 3.4 is provided in Figure 3.5. Obviously, this pattern definition is much more accessible to humans.

The XML structure of the complete configuration file is shown in Figure 3.6. Its hierarchy confers to the structure of pattern elements as it is described in Section 3.6 and shown in Figure 3.3.

The attributes `id` and `title`, present in the elements `segment` and its

```
...
<line name="Goal in Context" title="true" optional="true">
  <prefix>Goal in Context</prefix>
</line>

<line name="id" id="true">
  <prefix>Use\s+Case\s+UC_</prefix>
</line>

<segment name="Covered feature requests" optional="true">
  <prefix startline="true">Covered\s+feature\s+</prefix>
</segment>
...
```

Figure 3.5: Part of the XML configuration modelling the pattern that results in the regular expression as listed in Figure 3.4.

specialized version `line`, refer to the state of a segment - usually, one of the segments is a unique identifier that may be used for referencing the document, and another segment contains the title of the document.

The XML configuration file is mapped onto the objects mentioned above by an object mapping module called **Digester**. This very helpful module has been developed as part of the Jakarta project and is used by countless applications for reading in configuration files.

> Basically, the Digester package lets you configure an XML → Java object mapping module, which triggers certain actions called rules whenever a particular pattern of nested XML elements is recognized. [Pro06a]

3.7.2 The Digester object mapping

The rule setup for creating our object tree of pattern elements is shown in Figure 3.7. Four different types of rules are used:

1. An `ObjectCreate` rule triggers the Digester to create a new instance of the specified class and push it onto the stack.

2. `SetProperties` causes the Digester to initialize all properties of the stack's top object according to the attribute values provided in the triggering XML element by using the JavaBeans Introspection API.

3. `CallMethod` triggers the call of a method with the content of the triggering XML node provided as input parameter.

```
<!DOCTYPE doctypes [

  <!ELEMENT doctypes (doctype+)>
  <!ELEMENT doctype (textpattern)>
  <!ELEMENT textpattern(removelist,
                        delimiter,
                        (segment|line)+)>
  <!ELEMENT removelist(remove+)>
  <!ELEMENT delimiter(#PCDATA)>
  <!ELEMENT segment (prefix)>
  <!ELEMENT line (prefix)>
  <!ELEMENT prefix (#PCDATA)>

  <!ATTLIST doctype
      name CDATA #REQUIRED
      type CDATA #REQUIRED
      directory (true|false) "false"
  >

  <!ATTLIST segment
      name CDATA #REQUIRED
      id (true|false) "false"
      title (true|false) "false"
      optional (true|false) "false"
  >

  <!ATTLIST line
      name CDATA #REQUIRED
      id (true|false) "false"
      title (true|false) "false"
      optional (true|false) "false"
  >

  <!ATTLIST prefix
      line (true|false) "false"
      startline (true|false) "false"
      maystartline (true|false) "false"
  >
]>
```

Figure 3.6: Document type definition (DTD) of the doctypes configuration file.

```
digester.push(this);

digester.addObjectCreate("*/doctype",
                         "at.tuwien.serum.model.DocType");
digester.addSetProperties("*/doctype");

// Textpattern
digester.addObjectCreate("*/textpattern",
                         "at.tuwien.serum.model.TextPattern");
digester.addCallMethod("*/delimiter","setDelimiter",0);
digester.addCallMethod("*/remove","addRemove",0);

    // Line
digester.addObjectCreate("*/line",
                         "at.tuwien.serum.model.Line");
digester.addSetProperties("*/line");

        // Prefix
digester.addObjectCreate("*/prefix",
                         "at.tuwien.serum.model.Prefix");
digester.addSetProperties("*/prefix");
digester.addCallMethod("*/prefix","setContent",0);
digester.addSetNext("*/prefix","setPrefix");

        // Body
digester.addCallMethod("*/body","setBody",0);

    // end Line
digester.addSetNext("*/line","addSegmentPattern");

    // Segment
digester.addObjectCreate("*/segment",
                         "at.tuwien.serum.model.SegmentPattern");
digester.addSetProperties("*/segment");

    // end Segment
digester.addSetNext("*/segment","addSegmentPattern");

// end textpattern
digester.addSetNext("*/textpattern","setTextPattern");

// end doctype
digester.addSetNext("*/doctype","addDocType");

digester.parse(file);
```

Figure 3.7: Rule setup for the Digester component.

4. `SetNext` causes the Digester to use the object on top of the stack as a parameter to a method call that is executed on the following object on the stack.

All of these rules are triggered when an XML parser event matches an XML path specified by the *XPath*[4] expression that is provided in the rule definition, e. g. `*/textpattern`.

The digester component operates on an object stack, each action is applied to the object on top of the stack. The last rule of the setup, applying to the end of the root element in the XML file, thus causes the Digester to set the root element of the object hierarchy, containing the complete object tree, to the calling object that was pushed onto the stack before the `parse()` method was executed.

3.7.3 The Extractor

To initiate the training process, SERUM needs an open project, a document type, and a source to import from. When working with the user interface, the user will usually employ an iterative two-way approch:

1. Define and edit the document type pattern file and load the document types into the SERUM workspace.
2. Initiate the extraction process and review the results achieved with the current pattern.
3. If the user is satisfied with the results, he will save the extracted documents. If not, he may edit the pattern definition, reload it and initiate the extraction again until the results are satisfying.

The extraction algorithm itself works in a top-down manner from matching the patterns of complete documents to the small patterns of prefix and body inside the segments. The sequence diagram in Figure 3.8 shows the most important classes collaborating in the implementation; some steps and classes were omitted for clarity. The extraction starts with splitting the complete input string into a list of strings representing *candidate documents* according to the delimiter specified in the text pattern.

These candidate strings are then matched to the text pattern, and for each match, a new `Doc` is created by the `TextPattern` in the method `makeText`. The corresponding sequence diagram detailing this creation is provided in Figure 3.9, again with several steps omitted.

[4]Information about the XML Path Language can be found at `http://www.w3.org/TR/xpath`.

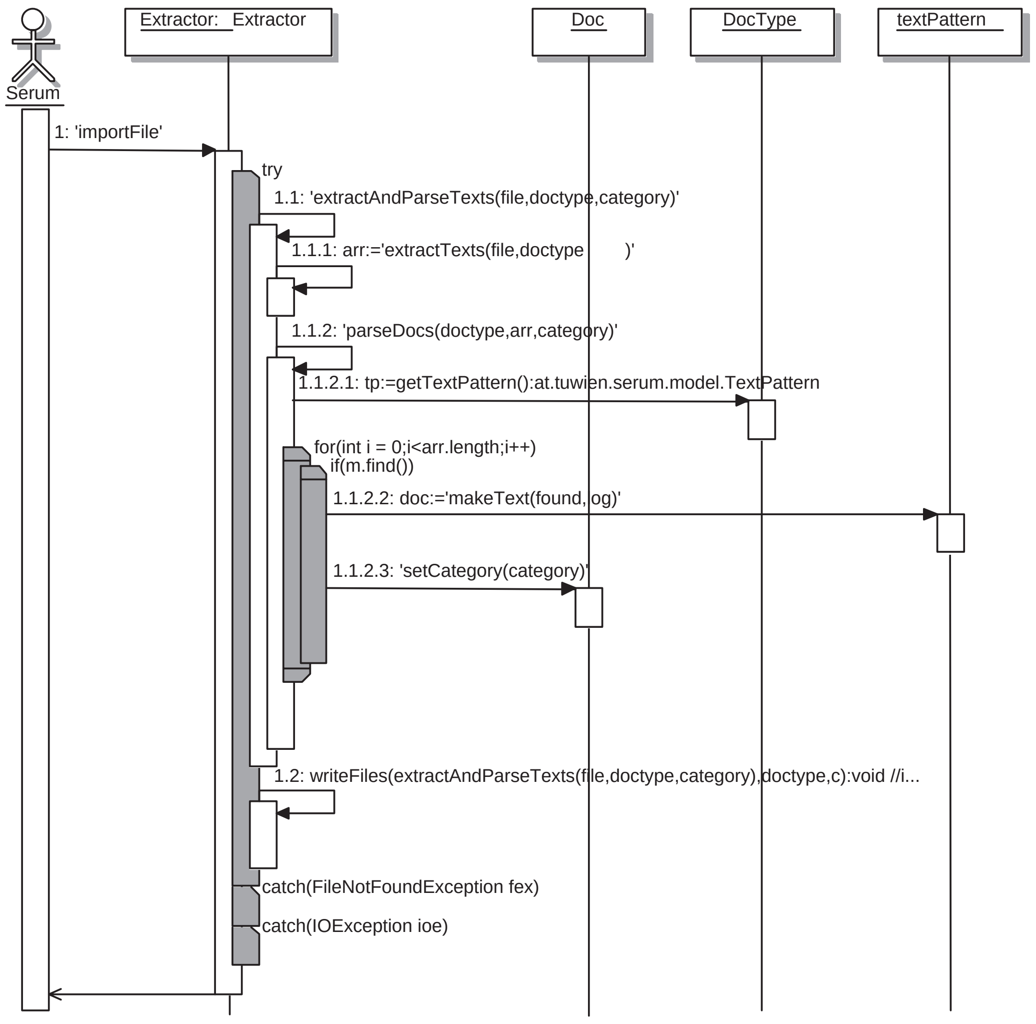

Serum
Extractor: Extractor
Doc
DocType
textPattern
1: 'importFile'
try
1.1: 'extractAndParseTexts(file,doctype,category)'
1.1.1: arr:='extractTexts(file,doctype)'
1.1.2: 'parseDocs(doctype,arr,category)'
1.1.2.1: tp:=getTextPattern():at.tuwien.serum.model.TextPattern
for(int i = 0;i<arr.length;i++)
if(m.find())
1.1.2.2: doc:='makeText(found, og)'
1.1.2.3: 'setCategory(category)'
1.2: writeFiles(extractAndParseTexts(file,doctype,category),doctype,c):void //i...
catch(FileNotFoundException fex)
catch(IOException ioe)

Figure 3.8: Sequence diagram showing the main classes collaborating in the extraction process.

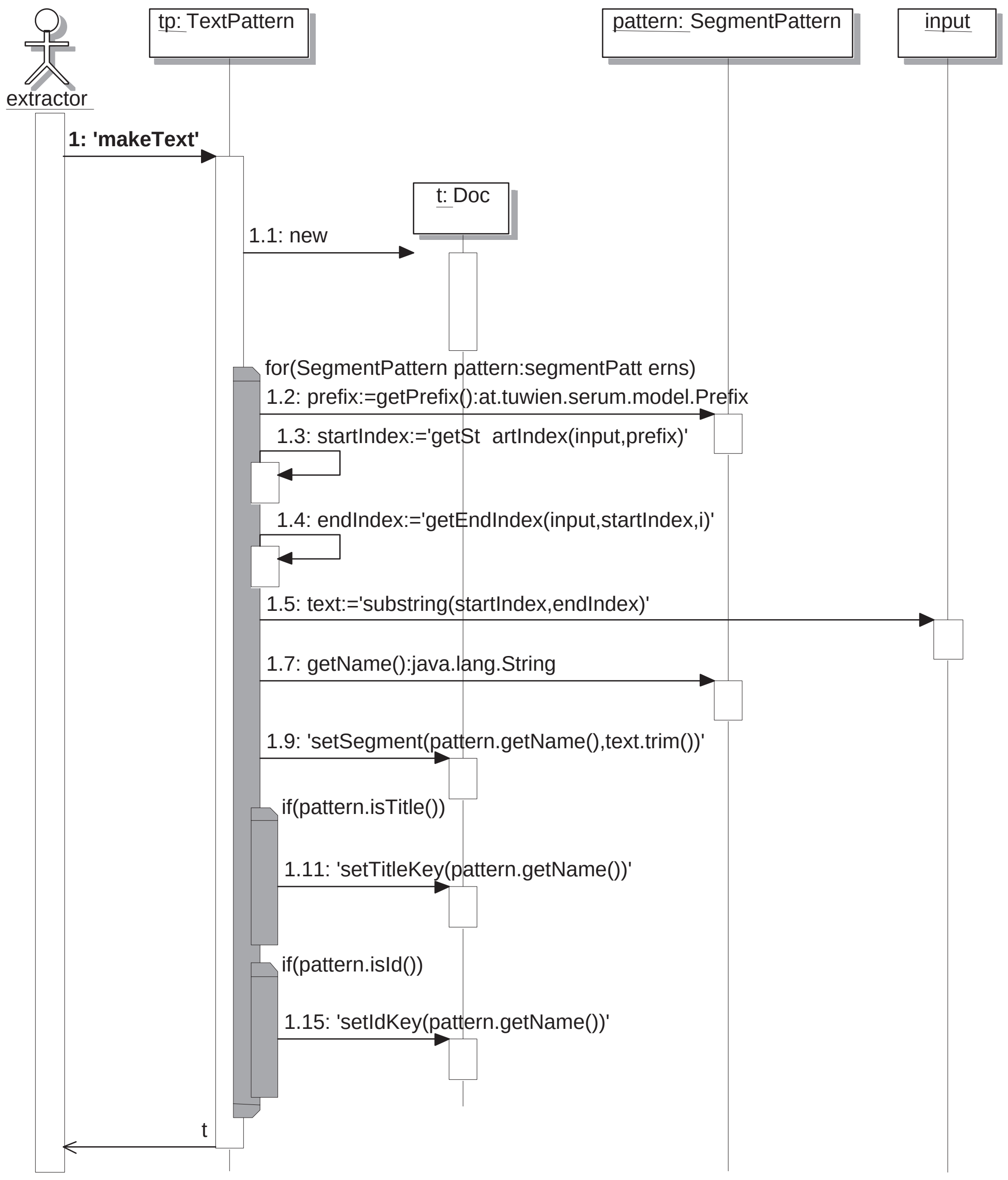

extractor
tp: TextPattern
pattern: SegmentPattern
input
1: 'makeText'
t: Doc
1.1: new
for(SegmentPattern pattern:segmentPatt erns)
1.2: prefix:=getPrefix():at.tuwien.serum.model.Prefix
1.3: startIndex:='getSt artIndex(input,prefix)'
1.4: endIndex:='getEndIndex(input,startIndex,i)'
1.5: text:='substring(startIndex,endIndex)'
1.7: getName():java.lang.String
1.9: 'setSegment(pattern.getName(),text.trim())'
if(pattern.isTitle())
1.11: 'setTitleKey(pattern.getName())'
if(pattern.isId())
1.15: 'setIdKey(pattern.getName())'
t

Figure 3.9: Sequence diagram of `TextPattern.makeText()` showing the process of creating a `Doc` from an input string.

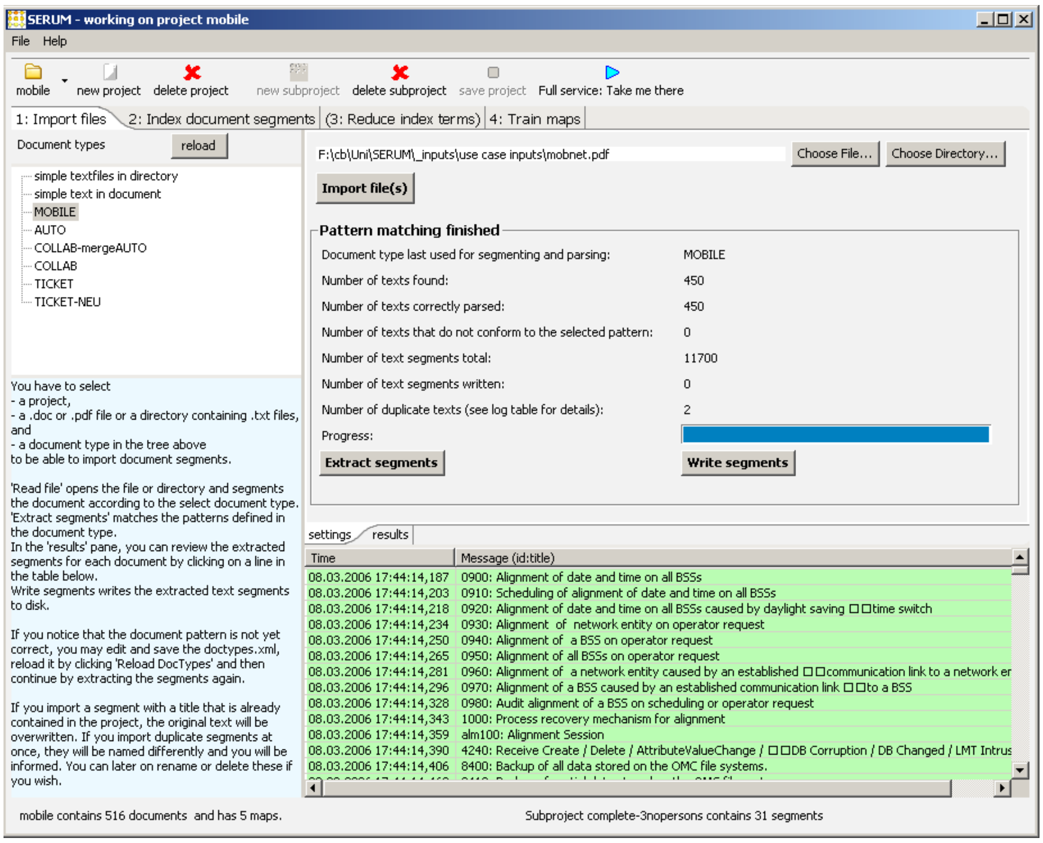
The `TextPattern` iterates over its `SegmentPatterns`, computes the boundaries of each segment and its prefix and body in the input string and initializes the segments from the according substrings. It also sets the keys to the `title` and `id` segments in the `Doc`.

After the `Doc` has been created and the call has returned to the `Extractor`, the category of the new `Doc` is set. After all Docs have been parsed, the resulting files are written to disk.

For each text that cannot be matched to the pattern, a notification is logged, and the GUI presents these texts to the user after finishing the extraction process. Furthermore, documents with duplicate IDs are logged and saved with altered IDs, and the user is warned about these events.

3.7.4 The user interface

Figure 3.10 shows the import screen of SERUM during the process of extraction.



F:\cb\Uni\SERUM_inputs\use case inputs\mobnet.pdf

You have to select
- a project,
- a .doc or .pdf file or a directory containing .txt files, and
- a document type in the tree above
to be able to import document segments.

'Read file' opens the file or directory and segments the document according to the select document type.
'Extract segments' matches the patterns defined in the document type.
In the 'results' pane, you can review the extracted segments for each document by clicking on a line in the table below.
Write segments writes the extracted text segments to disk.

If you notice that the document pattern is not yet correct, you may edit and save the doctypes.xml, reload it by clicking 'Reload DocTypes' and then continue by extracting the segments again.

If you import a segment with a title that is already contained in the project, the original text will be overwritten. If you import duplicate segments at once, they will be named differently and you will be informed. You can later on rename or delete these if you wish.

Pattern matching finished

Document type last used for segmenting and parsing:	MOBILE
Number of texts found:	450
Number of texts correctly parsed:	450
Number of texts that do not conform to the selected pattern:	0
Number of text segments total:	11700
Number of text segments written:	0
Number of duplicate texts (see log table for details):	2
Progress:	

Time	Message (id:title)
08.03.2006 17:44:14,187	0900: Alignment of date and time on all BSSs
08.03.2006 17:44:14,203	0910: Scheduling of alignment of date and time on all BSSs
08.03.2006 17:44:14,218	0920: Alignment of date and time on all BSSs caused by daylight saving □□time switch
08.03.2006 17:44:14,234	0930: Alignment of network entity on operator request
08.03.2006 17:44:14,250	0940: Alignment of a BSS on operator request
08.03.2006 17:44:14,265	0950: Alignment of all BSSs on operator request
08.03.2006 17:44:14,281	0960: Alignment of a network entity caused by an established □□communication link to a network er
08.03.2006 17:44:14,296	0970: Alignment of a BSS caused by an established communication link □□to a BSS
08.03.2006 17:44:14,328	0980: Audit alignment of a BSS on scheduling or operator request
08.03.2006 17:44:14,343	1000: Process recovery mechanism for alignment
08.03.2006 17:44:14,359	alm100: Alignment Session
08.03.2006 17:44:14,390	4240: Receive Create / Delete / AttributeValueChange / □□DB Corruption / DB Changed / LMT Intrus
08.03.2006 17:44:14,406	8400: Backup of all data stored on the OMC file systems.

mobile contains 516 documents and has 5 maps.

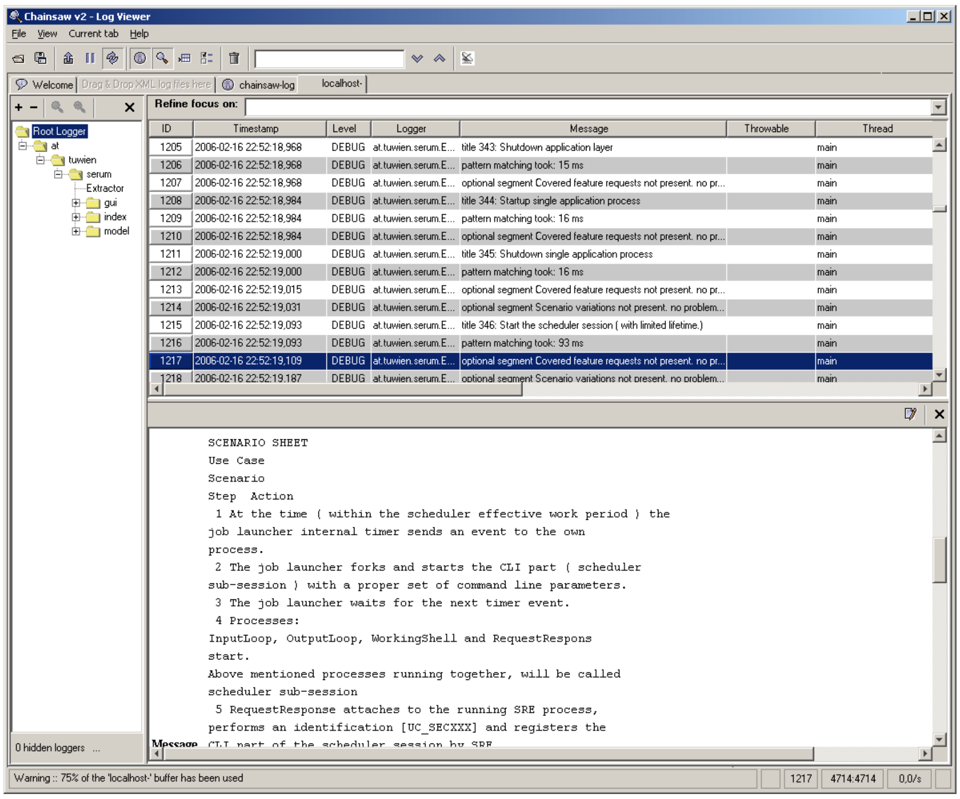
Subproject complete-3nopersons contains 31 segments

Figure 3.10: Import screen of SERUM.

In the toolbar, the user may select a project to work on, create a new one, etc.; in the upper left, he may select the document type conferring to the file he wants to import. The main part allows him to select a file or directory containing the documents he wants to extract, and initiate the extraction process according to the three steps of

1. Importing the file and splitting it into candidate documents,
2. Extracting these documents and their segments, and
3. Writing the extracted documents and segments to disk.

The user gets live feedback on the progress in the main panel as well as the preliminary results of document extraction in the table in the lower part of the window. If he wants even more detailed information about the pattern matching process, he may start the Chainsaw log viewer [Fou06b] and open a standard socket on his computer. SERUM will connect to this socket and directly send log messages to Chainsaw. A screenshot of Chainsaw during the extraction process is shown in Figure 3.11.

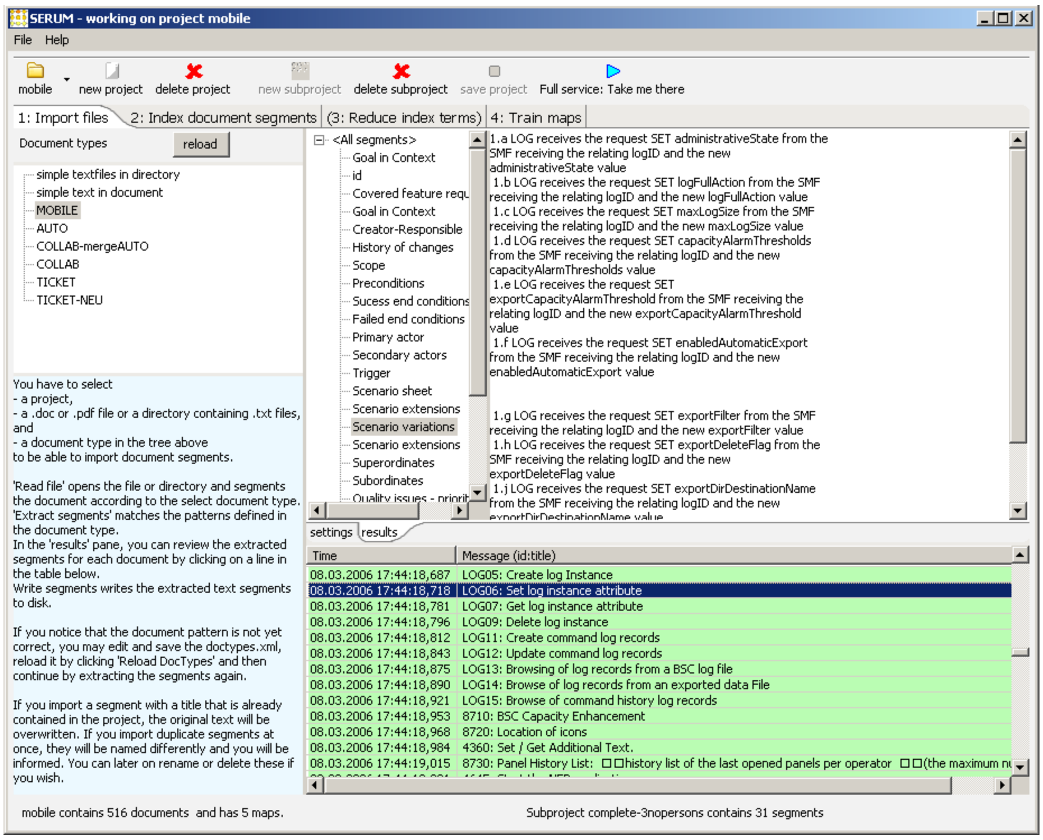


ID	Timestamp	Level	Logger	Message	Throwable	Thread
1205	2006-02-16 22:52:18,968	DEBUG	at.tuwien.serum.E...	title 343: Shutdown application layer		main
1206	2006-02-16 22:52:18,968	DEBUG	at.tuwien.serum.E...	pattern matching took: 15 ms		main
1207	2006-02-16 22:52:18,968	DEBUG	at.tuwien.serum.E...	optional segment Covered feature requests not present. no pr...		main
1208	2006-02-16 22:52:18,984	DEBUG	at.tuwien.serum.E...	title 344: Startup single application process		main
1209	2006-02-16 22:52:18,984	DEBUG	at.tuwien.serum.E...	pattern matching took: 16 ms		main
1210	2006-02-16 22:52:18,984	DEBUG	at.tuwien.serum.E...	optional segment Covered feature requests not present. no pr...		main
1211	2006-02-16 22:52:19,000	DEBUG	at.tuwien.serum.E...	title 345: Shutdown single application process		main
1212	2006-02-16 22:52:19,000	DEBUG	at.tuwien.serum.E...	pattern matching took: 16 ms		main
1213	2006-02-16 22:52:19,015	DEBUG	at.tuwien.serum.E...	optional segment Covered feature requests not present. no pr...		main
1214	2006-02-16 22:52:19,031	DEBUG	at.tuwien.serum.E...	optional segment Scenario variations not present. no problem...		main
1215	2006-02-16 22:52:19,093	DEBUG	at.tuwien.serum.E...	title 346: Start the scheduler session (with limited lifetime.)		main
1216	2006-02-16 22:52:19,093	DEBUG	at.tuwien.serum.E...	pattern matching took: 93 ms		main
1217	2006-02-16 22:52:19,109	DEBUG	at.tuwien.serum.E...	optional segment Covered feature requests not present. no pr...		main
1218	2006-02-16 22:52:19,187	DEBUG	at.tuwien.serum.E...	optional segment Scenario variations not present. no problem...		main

```
SCENARIO SHEET
Use Case
Scenario
Step  Action
 1 At the time ( within the scheduler effective work period ) the
job launcher internal timer sends an event to the own
process.
 2 The job launcher forks and starts the CLI part ( scheduler
sub-session ) with a proper set of command line parameters.
 3 The job launcher waits for the next timer event.
 4 Processes:
InputLoop, OutputLoop, WorkingShell and RequestRespons
start.
Above mentioned processes running together, will be called
scheduler sub-session
 5 RequestResponse attaches to the running SRE process,
performs an identification [UC_SECXXX] and registers the
CLI part of the scheduler session by SRE
```

```
Warning :: 75% of the 'localhost-' buffer has been used    1217    4714:4714    0,0/s
```

Figure 3.11: Chainsaw log during the extraction process.



You have to select
- a project,
- a .doc or .pdf file or a directory containing .txt files, and
- a document type in the tree above
to be able to import document segments.

'Read file' opens the file or directory and segments the document according to the select document type.
'Extract segments' matches the patterns defined in the document type.
In the 'results' pane, you can review the extracted segments for each document by clicking on a line in the table below.
Write segments writes the extracted text segments to disk.

If you notice that the document pattern is not yet correct, you may edit and save the doctypes.xml, reload it by clicking 'Reload DocTypes' and then continue by extracting the segments again.

If you import a segment with a title that is already contained in the project, the original text will be overwritten. If you import duplicate segments at once, they will be named differently and you will be informed. You can later on rename or delete these if you wish.

1.a LOG receives the request SET administrativeState from the SMF receiving the relating logID and the new administrativeState value
1.b LOG receives the request SET logFullAction from the SMF receiving the relating logID and the new logFullAction value
1.c LOG receives the request SET maxLogSize from the SMF receiving the relating logID and the new maxLogSize value
1.d LOG receives the request SET capacityAlarmThresholds from the SMF receiving the relating logID and the new capacityAlarmThresholds value
1.e LOG receives the request SET exportCapacityAlarmThreshold from the SMF receiving the relating logID and the new exportCapacityAlarmThreshold value
1.f LOG receives the request SET enabledAutomaticExport from the SMF receiving the relating logID and the new enabledAutomaticExport value

1.g LOG receives the request SET exportFilter from the SMF receiving the relating logID and the new exportFilter value
1.h LOG receives the request SET exportDeleteFlag from the SMF receiving the relating logID and the new exportDeleteFlag value
1.j LOG receives the request SET exportDirDestinationName from the SMF receiving the relating logID and the new

Time	Message (id:title)
08.03.2006 17:44:18,687	LOG05: Create log Instance
08.03.2006 17:44:18,718	LOG06: Set log instance attribute
08.03.2006 17:44:18,781	LOG07: Get log instance attribute
08.03.2006 17:44:18,796	LOG09: Delete log instance
08.03.2006 17:44:18,812	LOG11: Create command log records
08.03.2006 17:44:18,843	LOG12: Update command log records
08.03.2006 17:44:18,875	LOG13: Browsing of log records from a BSC log file
08.03.2006 17:44:18,890	LOG14: Browse of log records from an exported data File
08.03.2006 17:44:18,921	LOG15: Browse of command history log records
08.03.2006 17:44:18,953	8710: BSC Capacity Enhancement
08.03.2006 17:44:18,968	8720: Location of icons
08.03.2006 17:44:18,984	4360: Set / Get Additional Text.
08.03.2006 17:44:19,015	8730: Panel History List: □□history list of the last opened panels per operator □□(the maximum nu

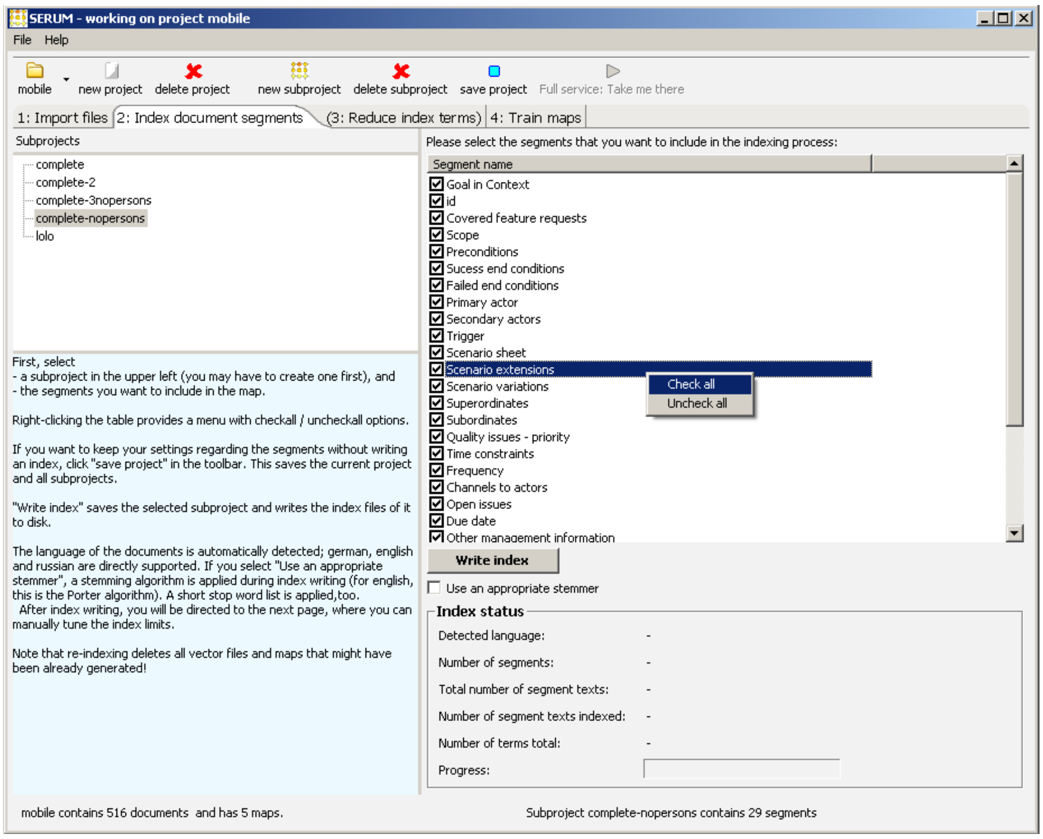
mobile contains 516 documents and has 5 maps.

Subproject complete-3nopersons contains 31 segments

Figure 3.12: Reviewing extraction results.

After the extraction process has finished, the user may view the results in the screen depicted in Figure 3.12.

By clicking on the table items and the segments in the upper tree, he can review the extraction results and decide if he wants to further optimize the pattern definition or return to the main panel and write the segments to disk.



Subprojects
- complete
- complete-2
- complete-3nopersons
- complete-nopersons
- lolo

First, select
- a subproject in the upper left (you may have to create one first), and
- the segments you want to include in the map.

Right-clicking the table provides a menu with checkall / uncheckall options.

If you want to keep your settings regarding the segments without writing an index, click "save project" in the toolbar. This saves the current project and all subprojects.

"Write index" saves the selected subproject and writes the index files of it to disk.

The language of the documents is automatically detected; german, english and russian are directly supported. If you select "Use an appropriate stemmer", a stemming algorithm is applied during index writing (for english, this is the Porter algorithm). A short stop word list is applied,too.
After index writing, you will be directed to the next page, where you can manually tune the index limits.

Note that re-indexing deletes all vector files and maps that might have been already generated!

Please select the segments that you want to include in the indexing process:

Segment name
✓ Goal in Context
✓ id
✓ Covered feature requests
✓ Scope
✓ Preconditions
✓ Sucess end conditions
✓ Failed end conditions
✓ Primary actor
✓ Secondary actors
✓ Trigger
✓ Scenario sheet
✓ Scenario extensions
✓ Scenario variations
✓ Superordinates
✓ Subordinates
✓ Quality issues - priority
✓ Time constraints
✓ Frequency
✓ Channels to actors
✓ Open issues
✓ Due date
✓ Other management information

☐ Use an appropriate stemmer

Index status

Detected language:	-
Number of segments:	-
Total number of segment texts:	-
Number of segment texts indexed:	-
Number of terms total:	-
Progress:	

mobile contains 516 documents and has 5 maps.

Subproject complete-nopersons contains 29 segments

Figure 3.13: Selection of segments to include in indexing.

3.8 Indexing

As soon as a project contains some documents, a map can be created to organize these into a 2-dimensional grid. The first steps are then to decide on which segments of the contained documents shall be relevant for organization and thus shall be used for indexing. This step is shown in Figure 3.13.

To cluster documents with the SOM, we need to represent them in a vector space model. The next step is creating an index, which can then be reduced by applying stop word filtering, defining percentage thresholds for word frequencies, etc. – this is initiated by the controls in the lower right of the screen, where the user can also decide whether to use a word stemming filter or not.

The next sections will describe the indexing component of SERUM and the feature reduction process.

3.8.1 Feature extraction: Indexing

SERUM relies on the indexing capabilities provided by the Apache Lucene framework. The SERUM `Indexer` adds documents to an `IndexWriter`, where each document consists of a series of *fields* like the title, the content, an ID, etc.; only the content is used for indexing.

Because we want to be able to cluster documents according to the similarity of specific sections as described above, we need to construct the *content* field of a document; moreover, we want to preserve information about the *category* a document belongs to, like the project name of a use case description. These services are provided by the `SerumDocument`, which dynamically concatenates the desired sections as configured by the user in the settings of the map, and provides this content to the `IndexWriter`. The latter saves the resulting index to a file that can be read by the `IndexReader`.

The `index()` method of the `Indexer`, which is depicted in Figure 3.14, serves two purposes.

1. It controls the index creation process of the Lucene `IndexWriter` component adapted from the Term Selection Tool (TeseT), and
2. it creates a `DocWriter` that writes each document contained in the map to an HTML file which can later be linked in the HTML output of a SOM and displayed in the SOMviewer.

The `IndexWriter` itself relies on an `Analyzer` that inspects each document and retrieves its word frequencies. This component can also be used for **preliminary feature reduction** steps, in particular for stemming words as described in Section 2.2.3, but also for stop word filtering. These methods are heavily language-dependent, it is necessary to take the language of each text into consideration before applying an appropriate algorithm.

To be able to do this automatically, we use the Java Text Categorization Library (JCTL) [Sol06], which provides a convenient service taking a string as input and returning the language of this string. This service is called inside the `AnalyzerFactory` at `createAnalyzer`, which is called before the indexing process actually starts, with the content of the first document to index as input. The analyzer factory then constructs an appropriate analyzer according to the returned language and the settings provided by the user, which determine if stemming and stop word filtering shall be applied.

The Lucene framework provides ready-to-use classes for english, german, and russian texts providing both stop word removal and word stemming filters; additional languages can easily be added and are likely to be supplied in future releases of the framework.

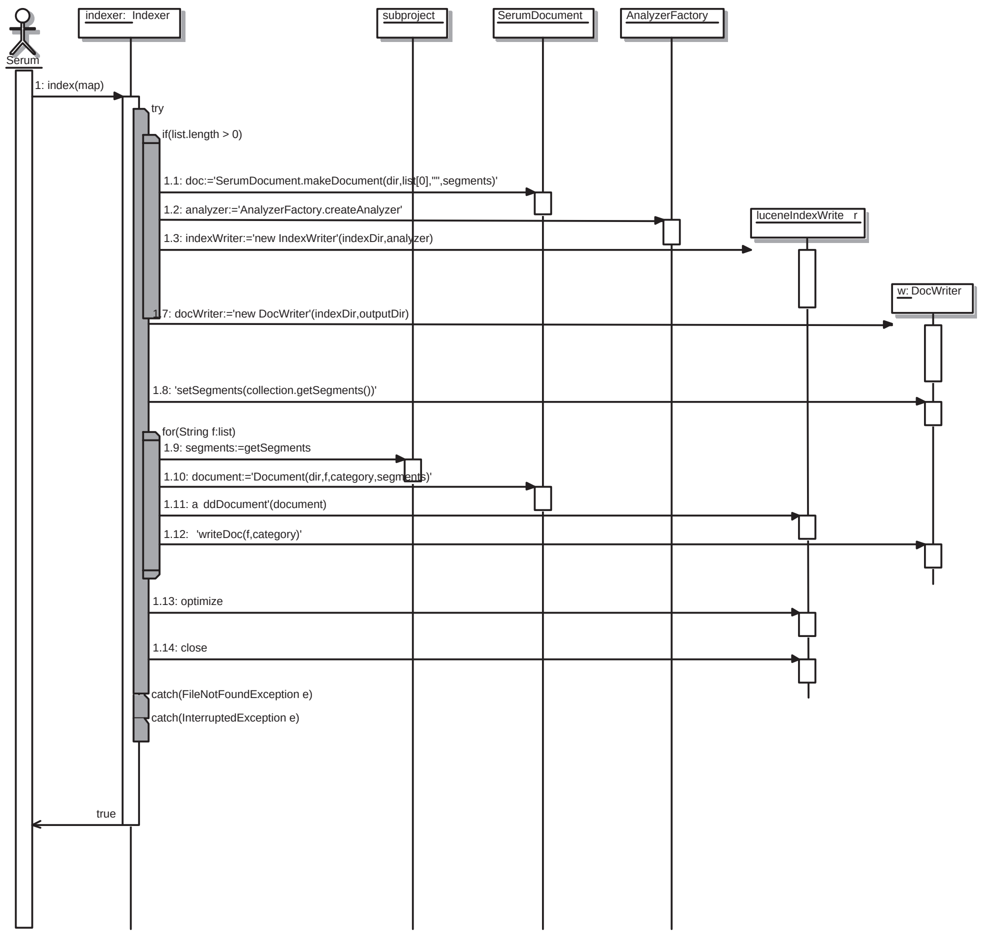

Serum
indexer: Indexer
subproject
SerumDocument
AnalyzerFactory
luceneIndexWriter
w: DocWriter
1: index(map)
try
if(list.length > 0)
1.1: doc:='SerumDocument.makeDocument(dir,list[0],"",segments)'
1.2: analyzer:='AnalyzerFactory.createAnalyzer'
1.3: indexWriter:='new IndexWriter'(indexDir,analyzer)
1.7: docWriter:='new DocWriter'(indexDir,outputDir)
1.8: 'setSegments(collection.getSegments())'
for(String f:list)
1.9: segments:=getSegments
1.10: document:='Document(dir,f,category,segments)'
1.11: a ddDocument'(document)
1.12: 'writeDoc(f,category)'
1.13: optimize
1.14: close
catch(FileNotFoundException e)
catch(InterruptedException e)
true

Figure 3.14: Sequence diagram showing the indexing of a subproject.

The `AnalyzerFactory` creates an `Analyzer` with the desired combination of filters and returns it to the `Indexer`, which then initializes the `DocWriter` component and runs the actual indexing and HTML writing process. This means adding all documents of the project to both `IndexWriter` and `DocWriter`, and finally closing the `IndexWriter`.

3.8.2 Pruning the feature space

The feature space pruning is carried out by an adapted version of the TeSeT component; this module provides feature space reduction by the following services:

1. Stop word filtering for english and german,
2. Removal of terms that match a regular expression,

3. Manual merging of terms through the user interface, and
4. Filtering of terms according to
   (a) their length, and
   (b) their frequency of occurence.

Because of the irregularities of natural language, the user will usually have to trade-off between an efficient workflow and optimal results. This means providing both an automatic feature reduction without manual interaction, as well as the option of manually fine-tuning the reduction parameters by adjusting the threshold settings, combining or merging terms, manually selecting terms for removal, and the like.

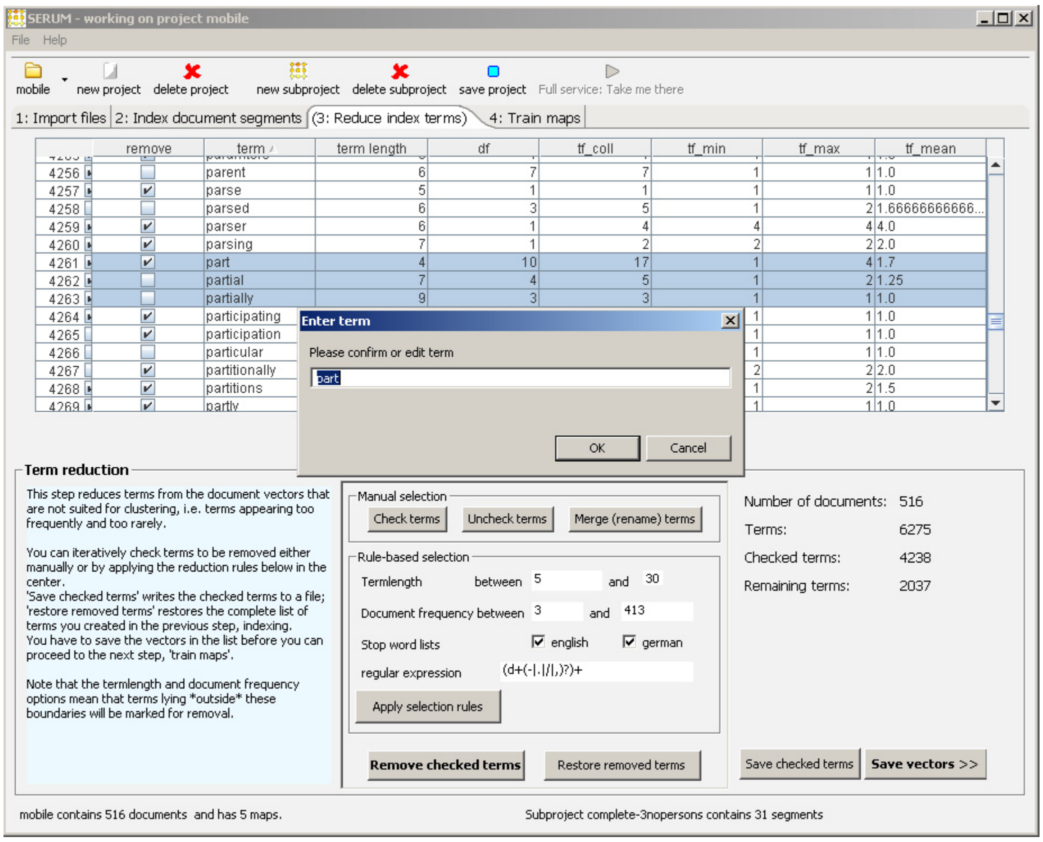
Table 3.7 provides the default values that are used for the reduction process if the user chooses automatic reduction; terms lying outside the given thresholds are marked for removal. These values are also preset in the reduction screen on activation, so the user can start at a reasonable point and further optimize it until the outcome meets his expectations. The user can adjust these default settings to his needs by editing the properties file `serum.prop` in SERUM's working directory.

The user interface of TeSeT is quite comfortable, a part of it is integrated into SERUM and shown in Figure 3.15. The user can manually fine-tune the settings or just decide to go along with the presets and proceed to map training. All removed terms are written to a text file that can be examined by the user.

An interesting and often useful option is the function to merge terms that denote the same semantic concept, yet cannot be automatically reduced to this concept with standard algorithmic approaches. For example, while the

Name	**Default Value**	**Description**
minDF	3	minimum *number* of documents a term must appear
maxDF	0.8	maximum *percentage* of documents a term may appear
mintermlength	5	the minimum length of a term
maxtermlength	30	the maximum length of a term
regex	(d+(-|.|/|,)?)+	a regular expression matching strings to be removed; the default value removes all numbers and dates given in common notations
stopDE	true	use german stop word list
stopEN	true	use english stop word list
stem	false	use appropriate stemming filter

Table 3.7: Default parameter settings used for feature space reduction.



	remove	term	term length	df	tf_coll	tf_min	tf_max	tf_mean
4256	☐	parent	6	7	7	1	1	1.0
4257	✓	parse	5	1	1	1	1	1.0
4258	☐	parsed	6	3	5	1	2	1.66666666666...
4259	✓	parser	6	1	4	4	4	4.0
4260	✓	parsing	7	1	2	2	2	2.0
4261	✓	part	4	10	17	1	4	1.7
4262	☐	partial	7	4	5	1	2	1.25
4263	☐	partially	9	3	3	1	1	1.0
4264	✓	participating				1	1	1.0
4265	✓	participation				1	1	1.0
4266	☐	particular				1	1	1.0
4267	✓	partitionally				2	2	2.0
4268	✓	partitions				1	2	1.5
4269	✓	partly				1	1	1.0

Enter term

Please confirm or edit term

part

OK Cancel

Term reduction

This step reduces terms from the document vectors that are not suited for clustering, i.e. terms appearing too frequently and too rarely.

You can iteratively check terms to be removed either manually or by applying the reduction rules below in the center.
'Save checked terms' writes the checked terms to a file; 'restore removed terms' restores the complete list of terms you created in the previous step, indexing.
You have to save the vectors in the list before you can proceed to the next step, 'train maps'.

Note that the termlength and document frequency options mean that terms lying *outside* these boundaries will be marked for removal.

Manual selection: Check terms | Uncheck terms | Merge (rename) terms

Rule-based selection

Termlength	between	5	and	30
Document frequency between		3	and	413
Stop word lists		[✓ english]		[✓ german]
regular expression		(d+(-|.|/|,)?)+		

Apply selection rules

Remove checked terms | Restore removed terms

Number of documents:	516
Terms:	6275
Checked terms:	4238
Remaining terms:	2037

Save checked terms | **Save vectors >>**

mobile contains 516 documents and has 5 maps.

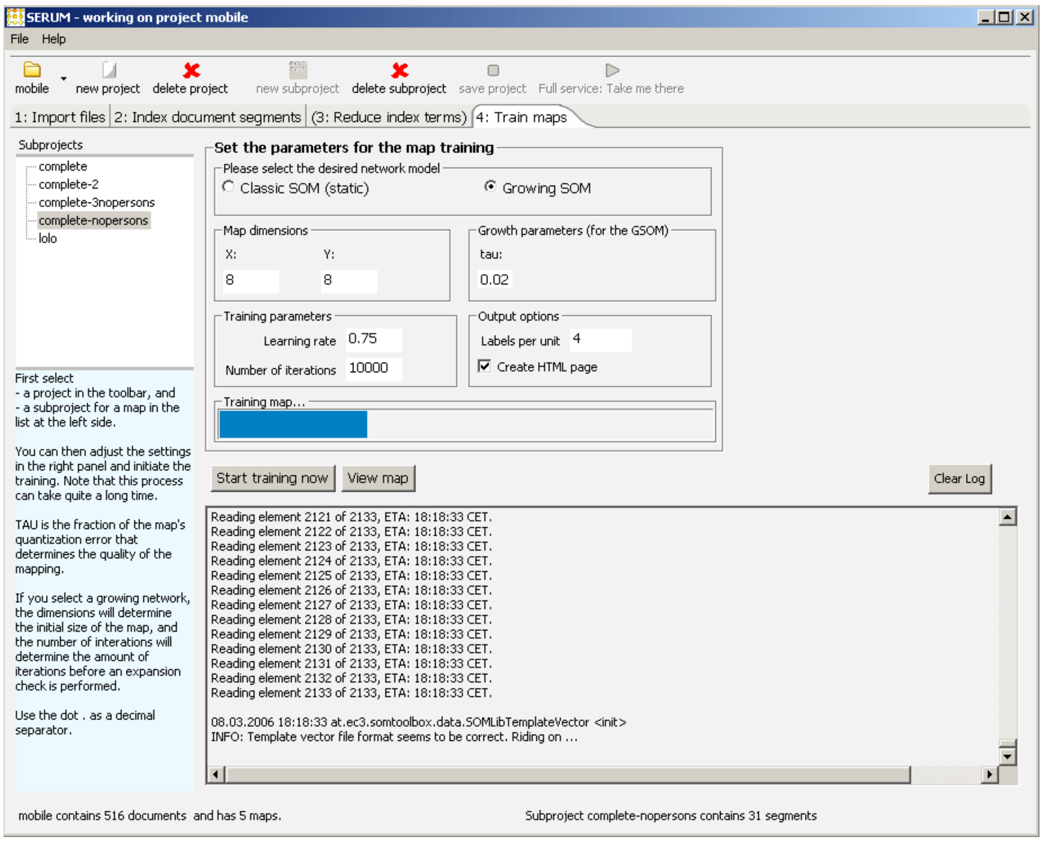
Subproject complete-3nopersons contains 31 segments

Figure 3.15: Feature space reduction screen while merging terms.

Porter stemmer would reduce the term *parts* to its root *part*, it would fail to do the same for words like *partial*. A human user can thus achieve much better results than a stemming algorithm in this regard; yet, he needs some time for this.

The performance of the original TeSeT component at the start of this project came out to be rather unsatisfying for larger collections; however, by optimizing the inner loops of the indexing classes during an early stage of development, the time needed for indexing could be reduced by more than 80 percent. This performance gain by a factor of six allows the user to repeatedly index and reduce document collections without experiencing significant idle times. Performance values measuring the time required to perform the various steps from extraction to map training are provided in Chapter 4.

When the user is finally satisfied with the results of the feature space reduction (or after the automatic reduction has taken place), a vector space



Subprojects
- complete
- complete-2
- complete-3nopersons
- complete-nopersons
- lolo

First select
- a project in the toolbar, and
- a subproject for a map in the list at the left side.

You can then adjust the settings in the right panel and initiate the training. Note that this process can take quite a long time.

TAU is the fraction of the map's quantization error that determines the quality of the mapping.

If you select a growing network, the dimensions will determine the initial size of the map, and the number of interations will determine the amount of iterations before an expansion check is performed.

Use the dot . as a decimal separator.

Set the parameters for the map training

Please select the desired network model: Classic SOM (static) [✓ Growing SOM]

Map dimensions: X: 8 Y: 8

Growth parameters (for the GSOM): tau: 0.02

Training parameters: Learning rate 0.75; Number of iterations 10000

Output options: Labels per unit 4; [✓] Create HTML page

Training map...

```
Reading element 2121 of 2133, ETA: 18:18:33 CET.
Reading element 2122 of 2133, ETA: 18:18:33 CET.
Reading element 2123 of 2133, ETA: 18:18:33 CET.
Reading element 2124 of 2133, ETA: 18:18:33 CET.
Reading element 2125 of 2133, ETA: 18:18:33 CET.
Reading element 2126 of 2133, ETA: 18:18:33 CET.
Reading element 2127 of 2133, ETA: 18:18:33 CET.
Reading element 2128 of 2133, ETA: 18:18:33 CET.
Reading element 2129 of 2133, ETA: 18:18:33 CET.
Reading element 2130 of 2133, ETA: 18:18:33 CET.
Reading element 2131 of 2133, ETA: 18:18:33 CET.
Reading element 2132 of 2133, ETA: 18:18:33 CET.
Reading element 2133 of 2133, ETA: 18:18:33 CET.

08.03.2006 18:18:33 at.ec3.somtoolbox.data.SOMLibTemplateVector <init>
INFO: Template vector file format seems to be correct. Riding on ...
```

mobile contains 516 documents and has 5 maps.   Subproject complete-nopersons contains 31 segments

Figure 3.16: SERUM's map training window.

model has to be written as input for the self-organizing map. This is the last step of preprocessing that is necessary before a map can be trained.

3.9 Map training

The last of the four panels in SERUM's user interface controls the map training process and provides live feedback on its progress to the user. It is depicted in Figure 3.16.

By default, SERUM suggests to train a growing self-organizing map (GSOM) with 10000 learning iterations between each expansion check and 4 labels assigned to each unit by the LabelSOM algorithm, and it further suggests to write an HTML output file.

The user can adjust these parameter settings for map training and initiate the training process. A progress bar informs him about the current state of training. Table 3.8 provides the values that are preset when the user enters

Name	Default Value	Description
model	growing	Use a growing self-organizing map
iterations	10000	number of iterations; for the growing map model, this parameter indicates the number of iterations before an expansion check is performed
tau	0.02	threshold that determines the desired data representation granularity and thus controls the growth process
learningrate	0.75	initial learningrate
x	3	initial number of columns in the map
y	3	initial number of rows in the map
labels	4	number of labels to generate for each unit
html	true	create an HTML output file for each map

Table 3.8: Default parameter settings used for map training.

the screen. Like the values provided in Table 3.7, these default settings can be changed by the user by editing the properties file `serum.prop`.

While the map grows, its current dimensions are updated in the user interface, and the user is notified with detailed status information in the log window below. The SOM Toolbox was extended with several progress listener interfaces to support this feedback.

The HTML output option allows users to view the resulting maps without needing a dedicated software; it is also convenient for publishing training results on the web.

This is the point which can also be reached directly from the first screen by selecting the “Full service” option in the upper right. The user may watch the application proceed through the successive steps until it stops here; the only information he has to provide is a name for the new map.

3.10 Interaction

After training a map, the user is ready to analyze it. The button “View map” starts a new SOMViewer application opening the currently active map in *document mode* – which means that the SERUM *DocViewer* component is activated instead of, e. g., the *PlaySOMPanel* that would be used with maps containing audio files. It is possible to start more than one SOMViewer in parallel to compare different maps.

The SOMViewer and DocViewer components will be discussed in detail in the next chapter.

```
<!DOCTYPE serum-jobs [
    <!ELEMENT serum-jobs (collection+)>
    <!ELEMENT collection (imports,maps)>
    <!ELEMENT imports (import+)>
    <!ELEMENT import EMPTY>
    <!ELEMENT maps (map+)>
    <!ELEMENT map (segments)>
    <!ELEMENT segments (segment+)>
    <!ELEMENT segment #PCDATA>

    <!ATTLIST collection name CDATA #REQUIRED>
    <!ATTLIST import doctype CDATA #REQUIRED
                     importPath CDATA #REQUIRED
                     category CDATA #REQUIRED>
    <!ATTLIST map name CDATA #REQUIRED
                  config CDATA>
]>
```

Figure 3.17: Document type definition of the batch input file.

3.11 Batch mode

If we want to process large amounts of text with similar characteristics – for example if we want import a large number of similar documents into a collection and train a number of maps with varying parameter settings to compare the outcomes – we may not want to do this by hand, no matter how comfortable the user interface is. Instead, we need a batch processing facility.

SERUM provides this by defining an XML input structure containing collections, map settings and import instructions that are followed by the batchworker component. Figure 3.17 provides the structural definition of the input XML file. This XML file is read by the `SerumWorker` in a similar way as the `doctypes.xml` discussed in Section 3.7.2 by using the Digester object mapping.

The settings in the XML file correspond to those discussed in the last sections; the attribute `config` of a map denotes the filename of a properties file containing the settings that should be used for map training; this is more convenient to provide these parameters, and also allows more than one map to share the same settings. A sample file that is used for batch processing is provided in appendix B. While processing this input, the batch worker logs its progress to the standard output as well as to the Chainsaw application, if the latter is active.

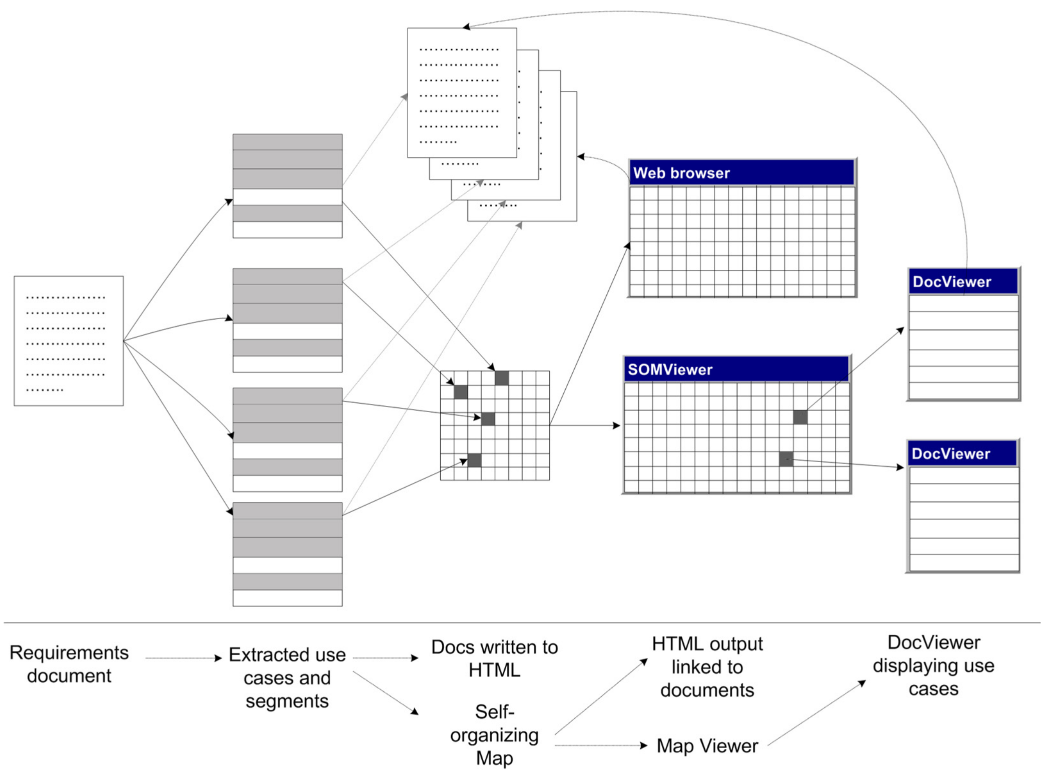


Figure 3.18: Way of a use case and its segments from extraction to interaction.

3.12 Summary

In this chapter, we discussed the tool SERUM – Self-oRganizing Use case Maps. We explained the need for such a tool, outlined the requirements that had to be met, and discussed its architecture, components and workflow in detail. Before moving on to presenting the resulting maps in the extended SOMViewer application, we would like to review the way a use case and its segments take from being extracted out of document files to being viewed by the user in the DocViewer. Figure 3.18 presents a diagram depicting this way.

The next chapter introduces the use case sets that we will use as a case study, and describes the extraction process, its parameters and results. We will then present the extended SOMViewer component with the DocViewer, and discuss the resulting maps.

Chapter 4

Extracting and Clustering Use Cases

In this chapter, we will use the tool presented in the last chapter to organize several collections of documents containing use case descriptions.

The chapter starts with an introduction to the use case sets we obtained from industrial partners. We will then describe the steps of importing, indexing, etc., and provide the parameters that were used. Finally, we present some of the resulting self-organizing maps and discuss the possibilities of exploring them in the SOMViewer as well as with a normal web browser. Both alternatives provide the option of viewing the original texts that have been mapped onto specific units on the map in the course of exploration.

4.1 The Use Case sets

Unsurprisingly, it is not trivial to get access to real-world instances of use cases in industrial environments. While several companies and institutions do actively apply use cases in describing user requirements (indeed, more than we initially expected), they are reluctant to give access to those use case sets, often even under terms of non-disclosure agreements. Similar experiences were indicated to us by several academics working in the field of use case applications.

In addition, some basic criteria had to be met by the use case sets, further narrowing the number of options for analysis:

- *Size.* A set of use cases should contain sufficient use cases to make content-based organization and clustering feasible. Several sets containing less than 20 use cases were dismissed.

- *Relevance.* A set should consist of real-world use cases, used in industrial environments, and not be artificially constructed. It should possibly involve large institutions with a software engineering track record, not just small software producers.

- *Structure.* The use cases should more or less conform to use case templates structured in some way, and not be totally informal text fragments describing some ill-defined system aspects.

- *Readiness.* The use cases should be ready to use with the techniques applied; it should not be necessary to prepare or substantially change the use cases – this would not be possible in a real-world environments, too.

We finally obtained access to four suitable sets of use cases; they are described in the following. Company names and details had to be omitted due to the terms of confidentiality agreements.

- *Set 1: TICKET.* TicketLine is a ticket reservation system developed at the Vienna University of Technology, primarily by graduate students, for use in both graduate and undergraduate courses. Three main releases were developed over the years, each with state-of-the-art methods and terminology; release 3 used use cases as primary requirement description approach. This set contains 66 use cases in German.

- *Set 2: AUTO.* This set of 20 use cases in English describes a control application in the automotive industry. The original specification on which the use case set is based was provided by a very large, international car manufacturer. The use case description was developed at a large German software engineering institute.

- *Set 3: COLLAB.* This is a large collaboration and document management framework. Project partners in this joint effort included an international company that is a large producer of white goods, and a very large international company in the market of global telecommunication systems and equipment. This set contains 26 use cases (in English).

- *Set 4: MOBILE.* This large use case set describes the behavior of software functionality in the field of mobile communication. It was created at one of the largest international organizations operating in communication, power infrastructure, and electrical engineering (not the company involved in set 3). This set contains 450 use cases (in English).

Property	**TICKET**	**AUTO**	**COLLAB**	**MOBILE**
Domain	Administration	Automotive industry	Collaboration, document management	Mobile communication
Number of use cases	66	20	26	450
Document format	separate `txt` files in directory, each containing a single use case	single `pdf` containing all use cases	single `doc` containing all use cases	large 850-page `pdf` containing all use cases
Average use case size, in characters	1396	1054	657	1814
Standard deviation in use case size	460	365	244	642
Number of segments contained	7	11	8	25
Total number of segments	462	231	234	11700
Use case style	casual	fully dressed	medium (no detailed eventflow)	fully dressed
Complexity	low	medium	low	very high
Language	german	english	english	english

Table 4.1: Properties of the use case sets

Table 4.1 gives an overview of the four sets, along with the file formats, average use case size, variance of use case size in characters, and characteristics of the use case descriptions. The size of a single use case varies between one and three pages.

The segments contained in the use case sets varied greatly, as the organizations obviously follow very different guidelines and templates. Table 4.2 lists the segments contained in each of the four use case sets in their order of appearance in the use case descriptions.

4.2 Importing and indexing use cases

The four use case sets come in three different input forms, as Table 4.1 shows.

1. The use case descriptions in the TICKET set are already split up into text files, where each file contains one use case description;
2. Descriptions in the sets AUTO and MOBILE were delivered as part of

TICKET	**AUTO**	**COLLAB**	**MOBILE**
Name	Full name	Name	Goal in Context
Summary	Actors	Actors	ID
Preconditions	Intent	Goal	Covered feature requests
Eventflow	Preconditions	Includes	Creator and Responsible
Success conditions	Event flow	Triggers	History of Changes
Non-functional requirements	Exceptions	Preconditions	Scope
Notes	Rules	Success end conditions	Preconditions
	Quality constraints	Failed end conditions	Success end conditions
	Monitored environment variables		Failed end conditions
	Controlled environment variables		Primary actor
	Post conditions		Secondary actors
			Trigger
			Scenario sheet
			Scenario extensions
			Scenario variations
			Superordinates
			Subordinates
			Quality issues
			Priority
			Time constraints
			Frequency
			Channels to actors
			Open issues
			Due date
			Other management information
7 segments	11 segments	8 segments	25 segments

Table 4.2: Segments contained in each use case set

a requirements document in `pdf` format; and

3. COLLAB use cases were contained in a `doc` file, again within a requirements document.

4.2.1 Pattern definition

The `doctypes.xml` is provided in full length in Appendix A. We will here only discuss a few short examples of segment definitions that illustrate typical requirements in practice. These are

- Inconsistent section headings,
- missing sections, and
- inconsistent ordering of sections.

Inconsistent section headings

The segment definition

```
<segment name="eventflow">
  <prefix startline="true">((Flow of events)|(Description)).</prefix>
</segment>
```

targets a common problem: People inevitably make small, accidental errors when writing headlines and texts – often typing errors, but also small confusions of terms, etc.

Furthermore, it brings to our attention that even in the largest projects, consistent usage of requirements management tools is obviously very rare. Dedicated requirements management tools would of course produce uniform section headings instead of naming a section inconsistently like *Flow of events* and *Description*. Only in the COLLAB document was the usage of section headings and the like completely consistent. This greatly facilitated pattern definition and speeded up the import process considerably.

Similar problems of inconsistency are present in other sets, e. g. MOBILE, as can be seen in the following definition.

```
<segment name="Channels to actors">
  <prefix startline="true">((Channels)|(connections)) to\s+(actors)?
  </prefix>
</segment>
```

Missing sections

Especially in the MOBILE set, a lot of descriptions miss one or more sections. This leads to most of the sections being classified as *optional*, which of course complicates string matching. Nevertheless, this is a minor problem; the performance is still satisfying, as we will se later.

Mixed ordering of sections

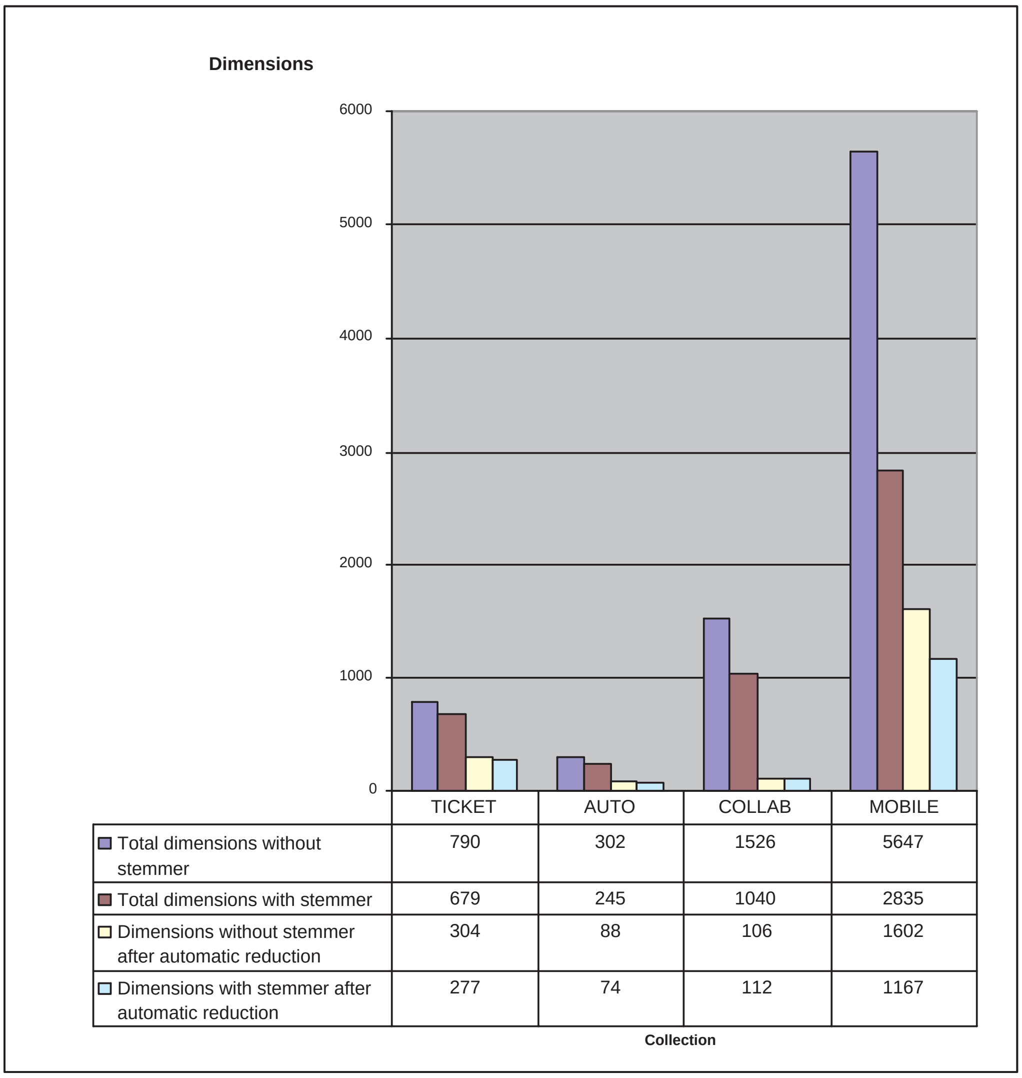
Like the problem of missing sections, the disorder of segments is primarily present in the largest and most complex use case set, MOBILE. It is solved simply by declaring the disordered element twice, as can be seen in the following fragment of the corresponding doctype. *Goal in context* is defined as first *and* as fourth element, because it does not always occur at the same place in the use case descriptions.

```
<delimiter>((USE CASE)|(Use Case)) DESCRIPTION \r?\n?</delimiter>
<!--  sequence of segment patterns -->

<line name="Goal in Context" title="true" optional="true">
  <prefix>Goal in Context</prefix>
</line>

<line name="id" id="true">
    <prefix>Use\s+Case\s+UC_</prefix>
</line>

<segment name="Covered feature requests" optional="true">
  <prefix startline="true">Covered\s+feature\s+</prefix>
</segment>

<line name="Goal in Context" title="true" optional="true">
  <prefix maystartline="true">Goal in Context</prefix>
</line>

<segment name="Creator-Responsible">
    <prefix line="true">Creator(s)?\s+(Responsible\s)?</prefix>
</segment>
```

The fragment above also illustrates another occurence of differing spelling, namely the `delimiter` in the first line. In several instances, the title text was written partly in lower case letters.

4.2.2 Feature space reduction

We applied several different reduction settings to each collection, varying parameters like the word frequency thresholds as well as comparing manual fine-tuning with the results achieved by automatic reduction. Figure 4.1 provides a comparison chart showing the dimensions, i. e. the number of distinct terms, of all four collections.



	TICKET	AUTO	COLLAB	MOBILE
🟪 Total dimensions without stemmer	790	302	1526	5647
🟥 Total dimensions with stemmer	679	245	1040	2835
🟨 Dimensions without stemmer after automatic reduction	304	88	106	1602
🟦 Dimensions with stemmer after automatic reduction	277	74	112	1167

Figure 4.1: Comparison chart for word dimensions with and without reductions.

Table 4.3 provides an excerpt of terms of the MOBILE documents prior

No stemmer	Porter stemmer
common	common
communication	comun
communications	–
communicationsalarm	communicationsalarm
compact	compact
compared	compar
compares	–
compatibility	compat
compatible	–
competence	compet
competenceflag	competenceflag
compl	compl
complete	–
completed	complet
completely	–
completeness	–
completion	–
complex	complex
(total: 6354)	(total: 2835)

Table 4.3: Terms in the MOBILE set with and without stemming.

to feature reduction. The left column displays terms identified without stemming, the right column lists terms retrieved by an analyzer using the Porter stemmer. Terms on both sides have been filtered with the same short english stop-word list.

Obviously, stemming reduces the feature space by a large factor and does a good job in unifying terms belonging to the same root. However, there are terms which cannot be stemmed automatically. The drawback of stemming is two-fold: First, it introduces language dependency; second, the labelling on the map can become more difficult to read. For example, a label reading *compatibility* would be more readable for a user exploring a map than the corresponding root term *compat*. When working with maps, the user thus may have to choose between optimized feature space reduction and optimized labelling.

Manual fine-tuning of the MOBILE feature space after analyzing it with a Porter stemmer took about 28 minutes; 834 terms remained. The changes that were made to the automatic presets were mainly of one of the following kinds:

- Merge semantically equivalent terms like *min*,*minim*, and *minimum*;
- Include terms that were marked for their removal due to their shortness, e. g. *log*, *unix*, *valu*, etc.;
- Merge typing errors like *objet* to *object*;
- Include abbreviations occuring quite often that obviously represent important concepts of a target domain, but were marked for removal because of their length;
- Remove common terms. Longer stop word lists would do the same task, but at the cost of increasing the probability that important terms are removed.

Table 4.4 provides some examples for removed terms using manual and automatic reduction, both with a stemmer. The column *df* gives the *document frequency* of the removed terms, i. e. the number of documents it appears in.

manual reduction	*df*	automatic reduction	*df*
abi	2	abi	2
		abl	19
abnorm	2	abnorm	2
abortact	1	abortact	1
abortalarmalign	1	abortalarmalign	1
aborterror	1	aborterror	1
abov	9	abov	9
absenc	1	absenc	1
absolut	3		
abstract	1	abstract	1
acc	2	acc	2
accordingli	1	accordingli	1
accumul	1	accumul	1
		ack	13
ackactivealarm	1	ackactivealarm	1
acknowldeg	1	acknowldeg	1
acknowledgeactivealarm	1	acknowledgeactivealarm	1
		...	
		dbm	204
		nma	21
		nmc	45

Table 4.4: Removed terms in the MOBILE set with manual and automatic reduction after stemming.

To get an idea of the effects of automatic reduction and compare the quality of the input mapping with the results of manual fine-tuning, we ran a series of tests with the MOBILE set and compared the outcomes of automatic versus manual reduction in terms of the Mean Quantization Error (MQE) and the Mean MQE (MMQE) of some resulting maps. Table 4.5 provides the results, which are also depicted in Figure 4.2.[1] Interestingly, automatic reduction produced lower MQE values than manual reduction, while removing fewer terms.

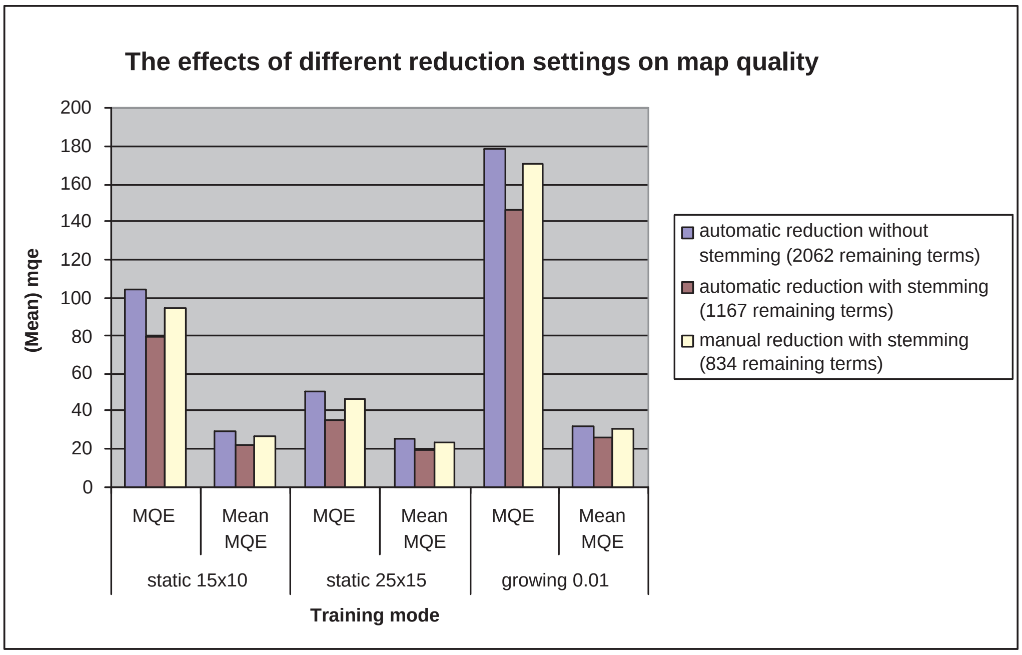


Figure 4.2: Chart comparing the effects of reduction settings on map quality.

[1]The static maps were trained with 50000 iterations, the growing maps with 10000 iterations between each expansion, starting with 5x5 units; the initial learning rate in both modes was set to 0.75. The resulting dimensions are provided below the MQE values in Table 4.5.

Reduction mode	**Dimensions before | after reduction**	**Static SOM size = 15x10 MQE | MMQE**	**Static SOM size = 25x15 MQE | MMQE**	**Growing SOM τ = 0.01 MQE | MMQE**
Automatic reduction without stemming	5647 | 1602	104.6 | 29.69	50.52 | 25.83	178.22 | 32.45 (size: 9x9)
Automatic reduction with stemming	2835 | 1167	79.01 | 22.32	34.73 | 19.34	146.08 | 26.04 (size: 9x9)
Manual reduction with stemming	2835 | 834	94.74 | 26.96	46.61 | 23.56	170.28 | 31.19 (size: 10x8)

Table 4.5: Effect of varying feature reduction settings on map quality

4.3 The resulting maps

The last step in the workflow of SERUM is starting the SOMViewer to analyze the resulting maps. The following figures show the SOMViewer displaying the static 15x10 map created of the MOBILE dataset after stemming and automatic feature reduction.

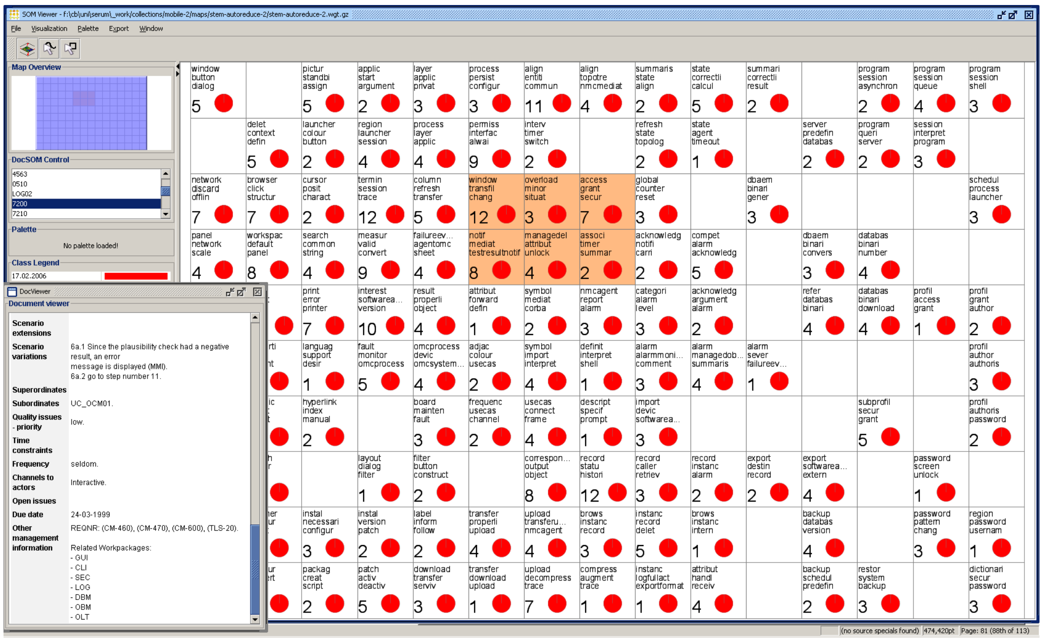
Figure 4.3 shows the full map of the MOBILE set, no enhanced visualization is active. The DocSOM control panel lists the names of all use cases that have been mapped onto the units currently selected on the map. The DocViewer panel shows the content of the use case with the name that is highlighted in the DocSOM panel.

If we turn on the smoothed data histograms visualization (SDH), we get a screen like that shown in Figure 4.4. This time, the zoom factor is higher; the color palette contains 256 gradients.

Figure 4.5 depicts another region of the same map with a 16-gradient grayscale D-Matrix visualization, and Figure 4.6 shows the HTML output of the MOBILE map in a web browser.

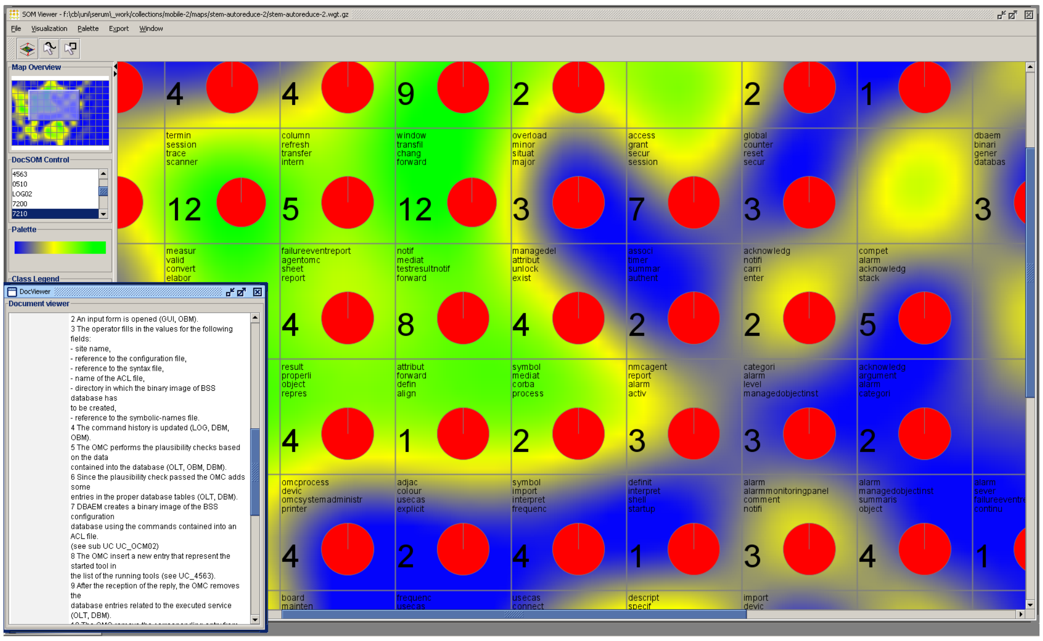
Due to confidentiality agreements, we cannot discuss all of the resulting maps in detail; only the TICKET set is available for closer examination. Consider the map depicted in Figure 4.7, which shows the use cases of the TICKET set. The visualization shows that occurences of the term *auffuehrung* tend to concentrate in the upper left corner. The use cases mapped onto the selected units as listed in the box on the left side accordingly contain use cases dealing with creating, editing, and cancelling events. But also mapped onto this region are use cases describing ticket reservation and ticket sales. Analyzing the use cases reveals high similarities of these descriptions.

Earlier experiments [ABRB05] revealed that the use case clustering is not simply determined by the main entity, like the *event* mentioned above. For example, use case descriptions dealing with search functionality are clustered



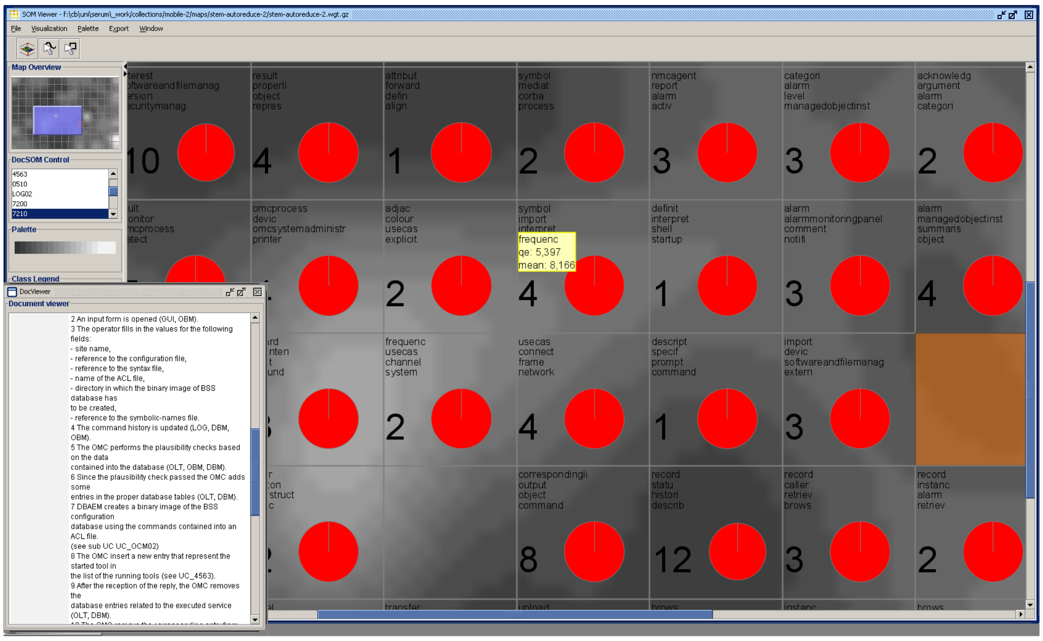
Scenario extensions	
Scenario variations	6a.1 Since the plausibility check had a negative result, an error message is displayed (MMI). 6a.2 go to step number 11.
Superordinates	
Subordinates	UC_OCM01.
Quality issues - priority	low.
Time constraints	
Frequency	seldom.
Channels to actors	Interactive.
Open issues	
Due date	24-03-1999
Other management information	REQNR: (CM-460), (CM-470), (CM-600), (TLS-20). Related Workpackages: - GUI - CLI - SEC - LOG - DBM - OBM - OLT

Figure 4.3: SomViewer showing the MOBILE static SOM.



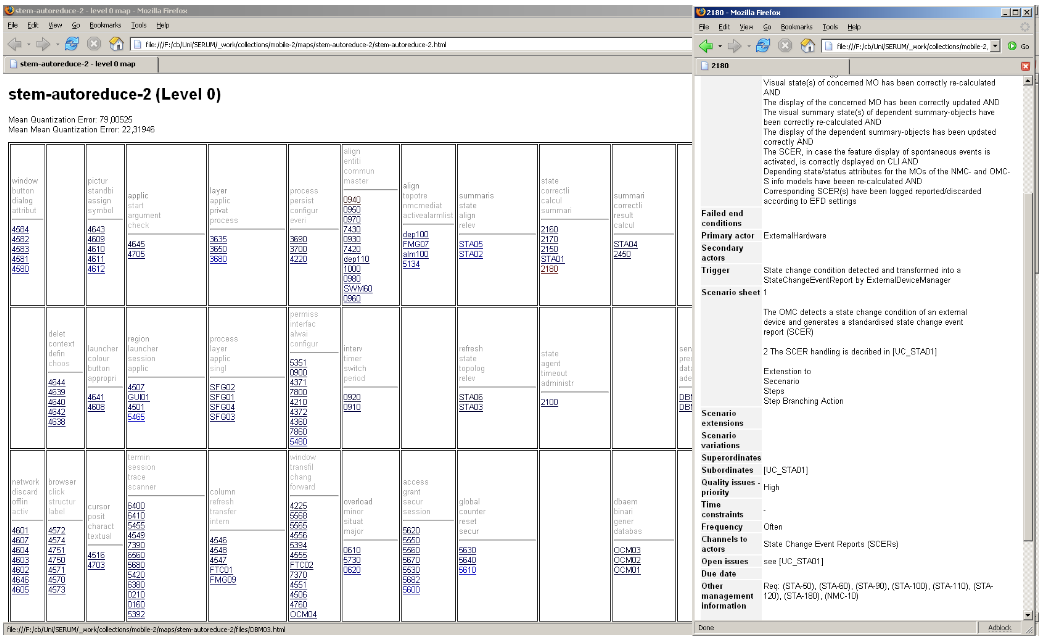
2 An input form is opened (GUI, OBM).
3 The operator fills in the values for the following fields:
- site name,
- reference to the configuration file,
- reference to the syntax file,
- name of the ACL file,
- directory in which the binary image of BSS database has
to be created,
- reference to the symbolic-names file.
4 The command history is updated (LOG, DBM, OBM).
5 The OMC performs the plausibility checks based on the data
contained into the database (OLT, OBM, DBM).
6 Since the plausibility check passed the OMC adds some
entries in the proper database tables (OLT, DBM).
7 DBAEM creates a binary image of the BSS configuration
database using the commands contained into an ACL file.
(see sub UC UC_OCM02)
8 The OMC insert a new entry that represent the started tool in
the list of the running tools (see UC_4563).
9 After the reception of the reply, the OMC removes the
database entries related to the executed service (OLT, DBM).

Figure 4.4: SOMViewer showing smoothed data histograms of the MOBILE SOM.


SOM Viewer - f:\cb\uni\serum_work\collections\mobile-2\maps\stem-autoreduce-2\stem-autoreduce-2.wgt.gz
File Visualization Palette Export Window
Map Overview
DocSOM Control
4563
0510
LOG02
7200
7210
Palette
Class Legend
terest oftwareandfilemanag ersion ecuritymanag 10
result properti object repres 4
attribut forward defin align 1
symbol mediat corba process 2
nmcagent report alarm activ 3
categori alarm level managedobjectinst 3
acknowledg argument alarm categori 2
ult onitor mcprocess etect
omcprocess devic omcsystemadministr printer
adjac colour usecas explicit 2
symbol import interpret
frequenc
qe: 5,397
mean: 8,166
4
definit interpret shell startup 1
alarm alarmmonitoringpanel comment notifi 3
alarm managedobjectinst summaris object 4
ird nten t und
frequenc usecas channel system 2
usecas connect frame network 4
descript specif prompt command 1
import devic softwareandfilemanag extern 3
r con struct c
correspondingli output object command 8
record statu histori describ 12
record caller retriev brows 3
record instanc alarm retriev 2
transfer
upload
brows
instanc
brows
DocViewer
Document viewer

2 An input form is opened (GUI, OBM).
3 The operator fills in the values for the following fields:
- site name,
- reference to the configuration file,
- reference to the syntax file,
- name of the ACL file,
- directory in which the binary image of BSS database has
to be created,
- reference to the symbolic-names file.
4 The command history is updated (LOG, DBM, OBM).
5 The OMC performs the plausibility checks based on the data
contained into the database (OLT, OBM, DBM).
6 Since the plausibility check passed the OMC adds some
entries in the proper database tables (OLT, DBM).
7 DBAEM creates a binary image of the BSS configuration
database using the commands contained into an ACL file.
(see sub UC UC_OCM02)
8 The OMC insert a new entry that represent the started tool in
the list of the running tools (see UC_4563).
9 After the reception of the reply, the OMC removes the
database entries related to the executed service (OLT, DBM).

Figure 4.5: SOMViewer with a D-matrix visualization of the MOBILE SOM.


stem-autoreduce-2 - level 0 map - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
file:///F:/cb/Uni/SERUM/_work/collections/mobile-2/maps/stem-autoreduce-2/stem-autoreduce-2.html
stem-autoreduce-2 - level 0 map

stem-autoreduce-2 (Level 0)

Mean Quantization Error: 79,00525
Mean Mean Quantization Error: 22,31946

window button dialog attribut 4584 4582 4583 4581 4580		pictur standbi assign symbol 4643 4609 4610 4611 4612	applic start argument check 4645 4705	layer applic privat process 3635 3650 3680	process persist configur everi 3690 3700 4220	align entiti commun master 0940 0950 0970 7430 0930 7420 dep110 1000 0980 SWM60 0960	align topotre nmcmediat activealarmlist dep100 FMG07 alm100 5134	summaris state align relev STA05 STA02	state correctli calcul summari 2160 2170 2150 STA01 2180	summari correctli result calcul STA04 2450	
	delet context defin choos 4644 4639 4640 4642 4638	launcher colour button appropri 4641 4608	region launcher session applic 4507 GUI01 4501 5465	process layer applic singl SFG02 SFG01 SFG04 SFG03	permiss interfac alwai configur 5351 0900 4371 7800 4210 4372 4360 7860 5480	interv timer switch period 0920 0910		refresh state topolog relev STA06 STA03	state agent timeout administr 2100		serv pred data ade DBI DBI
network discard offlin activ 4601 4607 4604 4603 4602 4646 4605	browser click structur label 4572 4574 4751 4750 4571 4570 4573	cursor posit charact textual 4516 4703	termin session trace scanner 6400 6410 5455 4549 7390 6560 5680 5420 6380 0210 0160 5392	column refresh transfer intern 4546 4548 4547 FTC01 FMG09	window transfil chang forward 4225 5568 5565 4556 5394 4555 FTC02 7370 4551 4506 4760 OCM04	overload minor situat major 0610 5730 0620	access grant secur session 5620 5550 5560 5670 5530 5682 5600	global counter reset secur 5630 5640 5610		dbaem binari gener databas OCM03 OCM02 OCM01	

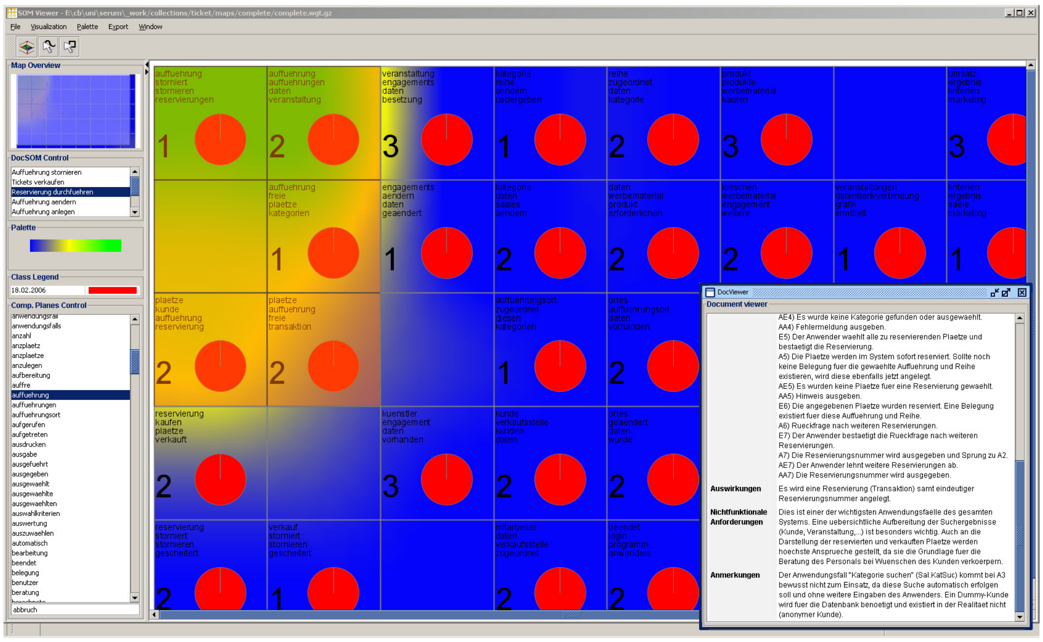
file:///F:/cb/Uni/SERUM/_work/collections/mobile-2/maps/stem-autoreduce-2/files/DBM03.html

2180 - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
file:///F:/cb/Uni/SERUM/_work/collections/mobile-2,
Go
2180

	Visual state(s) of concerned MO has been correctly re-calculated AND The display of the concerned MO has been correctly updated AND The visual summary state(s) of dependent summary-objects have been correctly re-calculated AND The display of the dependent summary-objects has been updated correctly AND The SCER, in case the feature display of spontaneous events is activated, is correctly dsplayed on CLI AND Depending state/status attributes for the MOs of the NMC- and OMC-S info models have beeen re-calculated AND Corresponding SCER(s) have been logged reported/discarded according to EFD settings
Failed end conditions	
Primary actor	ExternalHardware
Secondary actors	
Trigger	State change condition detected and transformed into a StateChangeEventReport by ExternalDeviceManager
Scenario sheet	1 The OMC detects a state change condition of an external device and generates a standardised state change event report (SCER) 2 The SCER handling is decribed in [UC_STA01] Extenstion to Secenario Steps Step Branching Action
Scenario extensions	
Scenario variations	
Superordinates	
Subordinates	[UC_STA01]
Quality issues - priority	High
Time constraints	-
Frequency	Often
Channels to actors	State Change Event Reports (SCERs)
Open issues	see [UC_STA01]
Due date	
Other management information	Req: (STA-50), (STA-60), (STA-90), (STA-100), (STA-110), (STA-120), (STA-180), (NMC-10)

Done

Figure 4.6: MOBILE map shown in a normal web browser.



AE4) Es wurde keine Kategorie gefunden oder ausgewaehlt.
AA4) Fehlermeldung ausgeben.
E5) Der Anwender waehlt alle zu reservierenden Plaetze und bestaetigt die Reservierung.
A5) Die Plaetze werden im System sofort reserviert. Sollte noch keine Belegung fuer die gewaehlte Auffuehrung und Reihe existieren, wird diese ebenfalls jetzt angelegt.
AE5) Es wurden keine Plaetze fuer eine Reservierung gewaehlt.
AA5) Hinweis ausgeben.
E6) Die angegebenen Plaetze wurden reserviert. Eine Belegung existiert fuer diese Auffuehrung und Reihe.
A6) Rueckfrage nach weiteren Reservierungen.
E7) Der Anwender bestaetigt die Rueckfrage nach weiteren Reservierungen.
A7) Die Reservierungsnummer wird ausgegeben und Sprung zu A2.
AE7) Der Anwender lehnt weitere Reservierungen ab.
AA7) Die Reservierungsnummer wird ausgegeben.

Auswirkungen	Es wird eine Reservierung (Transaktion) samt eindeutiger Reservierungsnummer angelegt.
Nichtfunktionale Anforderungen	Dies ist einer der wichtigsten Anwendungsfaelle des gesamten Systems. Eine uebersichtliche Aufbereitung der Suchergebnisse (Kunde, Veranstaltung,...) ist besonders wichtig. Auch an die Darstellung der reservierten und verkauften Plaetze werden hoechste Ansprueche gestellt, da sie die Grundlage fuer die Beratung des Personals bei Wuenschen des Kunden verkoerpern.
Anmerkungen	Der Anwendungsfall "Kategorie suchen" (Sal.KatSuc) kommt bei A3 bewusst nicht zum Einsatz, da diese Suche automatisch erfolgen soll und ohne weitere Eingaben des Anwenders. Ein Dummy-Kunde wird fuer die Datenbank benoetigt und existiert in der Realitaet nicht (anonymer Kunde).

Figure 4.7: SOMViewer showing the TICKET map.

in the lower right corner of the map, because the similarity of the search logic description is determining the clusters rather than the main entities of the use cases. Figure 4.8 show this cluster of use cases. The labels that have been assigned to the units illustrate the main focus of the mapped use case descriptions, as they deal partly with search criteria and user input (*suchkriterien, eingegeben, entsprechen*), and partly with the objects that are being searched (*kuenstler, reservierung, auffuehrungen, veranstaltungen*).

4.4 Viewing linked documents

With the DocViewer, it is possible to view the documents mapped onto units of the map directly within the SOMViewer. Figure 4.9 shows the DocViewer component in detail, displaying a MOBILE use case.

Within a web browser, the same map looks like shown in Figure 4.6. Of course, the information displayed is not nearly as helpful and rich as in the SOMViewer; but the map is readily distributable and easy to publish on the web. The documents themselves look exactly like in the DocViewer, because the latter displays the generated `html` that is also linked from the `html` output generated after map training.

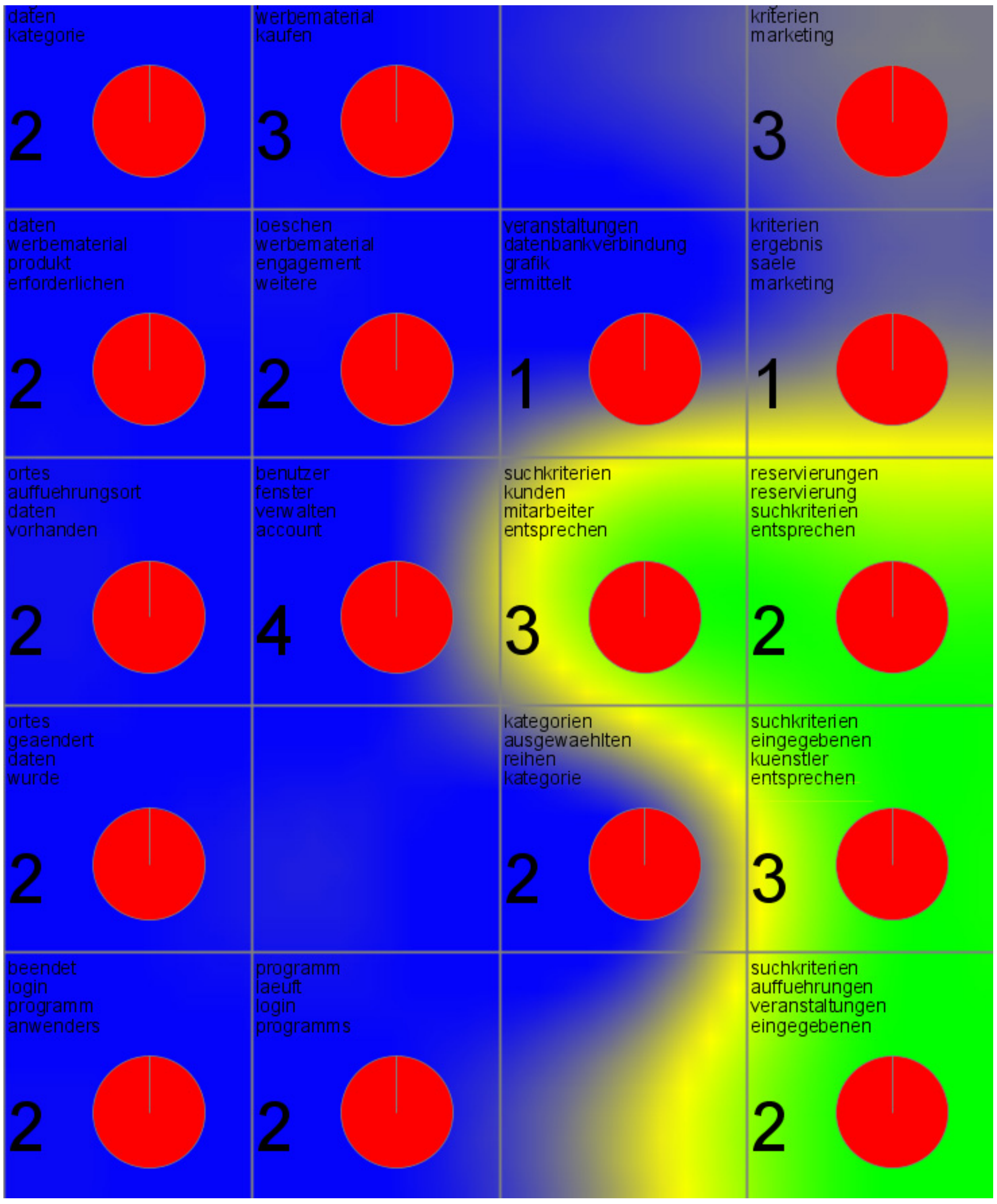
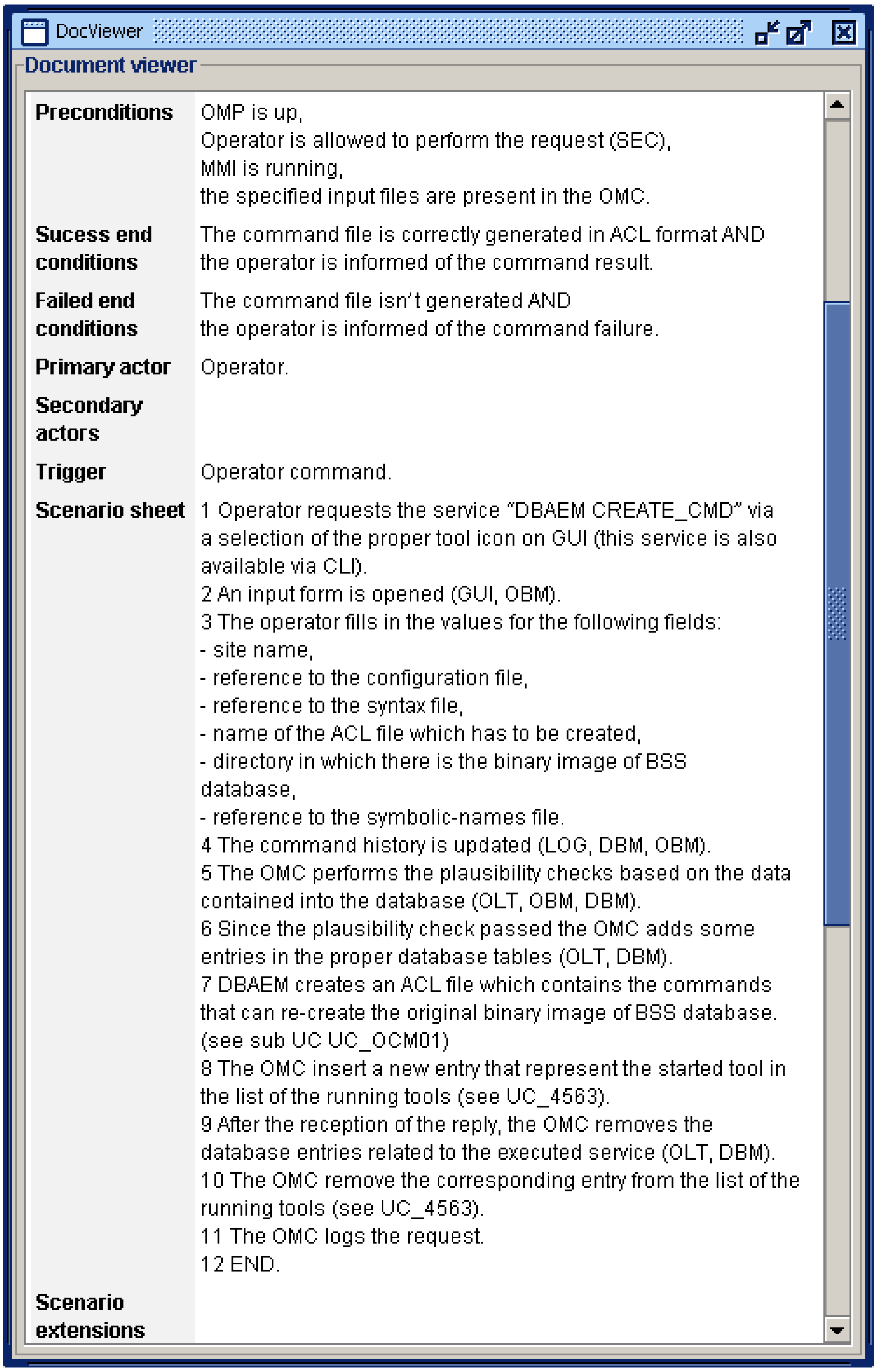


Figure 4.8: Detail of the SOMViewer showing the TICKET map.

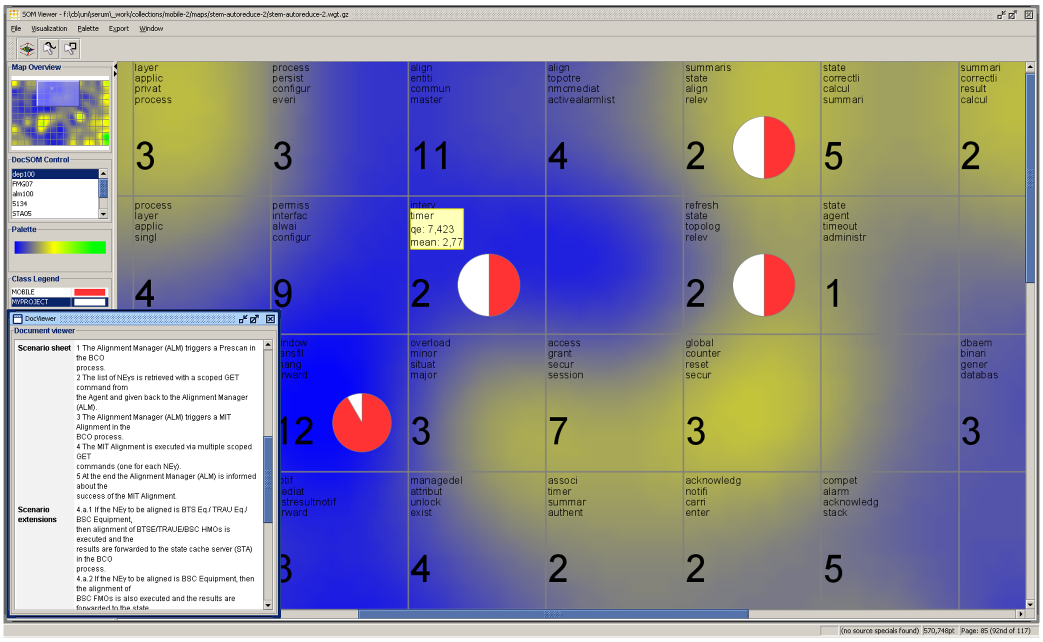


Preconditions	OMP is up, Operator is allowed to perform the request (SEC), MMI is running, the specified input files are present in the OMC.
Sucess end conditions	The command file is correctly generated in ACL format AND the operator is informed of the command result.
Failed end conditions	The command file isn't generated AND the operator is informed of the command failure.
Primary actor	Operator.
Secondary actors	
Trigger	Operator command.
Scenario sheet	1 Operator requests the service "DBAEM CREATE_CMD" via a selection of the proper tool icon on GUI (this service is also available via CLI). 2 An input form is opened (GUI, OBM). 3 The operator fills in the values for the following fields: - site name, - reference to the configuration file, - reference to the syntax file, - name of the ACL file which has to be created, - directory in which there is the binary image of BSS database, - reference to the symbolic-names file. 4 The command history is updated (LOG, DBM, OBM). 5 The OMC performs the plausibility checks based on the data contained into the database (OLT, OBM, DBM). 6 Since the plausibility check passed the OMC adds some entries in the proper database tables (OLT, DBM). 7 DBAEM creates an ACL file which contains the commands that can re-create the original binary image of BSS database. (see sub UC UC_OCM01) 8 The OMC insert a new entry that represent the started tool in the list of the running tools (see UC_4563). 9 After the reception of the reply, the OMC removes the database entries related to the executed service (OLT, DBM). 10 The OMC remove the corresponding entry from the list of the running tools (see UC_4563). 11 The OMC logs the request. 12 END.
Scenario extensions	

Figure 4.9: DocViewer component displaying a use case description.

4.5 Finding similar use cases: practical benefits

In earlier chapters, we mentioned that people in project organizations could benefit from their use case descriptions being organized in a SOM. So, how in particular can a single user benefit? Consider a situation where the use case descriptions of a new project are added to a collection already containing a lot of descriptions. The resulting map could look like that shown in Figure 4.10. The pie charts depict the shares of use cases from different projects in each unit; the class legend shows the corresponding category names, i. e. project names. By analyzing the surrounding use case descriptions in the neighborhood of each of the new use case descriptions, users might find out a lot about the project they are going to undertake.



SOM Viewer
File Visualization Palette Export Window
Map Overview
DocSOM Control
dep100
FMG07
alm100
5134
STA05
Palette
Class Legend
MOBILE 🟥
MYPROJECT ⬜
DocViewer
Document viewer
layer applic privat process 3
process persist configur everi 3
align entiti commun master 11
align topotre nmcmediat activealarmlist 4
summaris state align relev 2
state correcti calcul summari 5
summari correcti result calcul 2
process layer applic singl 4
permiss interfac alwai configur 9
timer qe: 7,423 mean: 2,77 2
refresh state topolog relev 2
state agent timeout administr 1
overload minor situat major 3
access grant secur session 7
global counter reset secur 3
dbaem binari gener databas 3
managedel attribut unlock exist 4
associ timer summar authent 2
acknowledg notifi carri enter 2
compet alarm acknowledg stack 5

Scenario sheet	1 The Alignment Manager (ALM) triggers a Prescan in the BCO process. 2 The list of NEys is retrieved with a scoped GET command from the Agent and given back to the Alignment Manager (ALM). 3 The Alignment Manager (ALM) triggers a MIT Alignment in the BCO process. 4 The MIT Alignment is executed via multiple scoped GET commands (one for each NEy). 5 At the end the Alignment Manager (ALM) is informed about the success of the MIT Alignment.
Scenario extensions	4.a.1 If the NEy to be aligned is BTS Eq./ TRAU Eq./ BSC Equipment, then alignment of BTSE/TRAUE/BSC HMOs is executed and the results are forwarded to the state cache server (STA) in the BCO process. 4.a.2 If the NEy to be aligned is BSC Equipment, then the alignment of BSC FMOs is also executed and the results are forwarded to the state

(no source specials found) 570,748pt Page: 85 (92nd of 117)

Figure 4.10: SOMViewer displaying a map with use cases from 2 different categories.

1. They will probably **discover use cases that describe similar functionality** in the vicinity of the new use cases. This intuitive neighbor discovery provides a much easier and thorough understanding than the user could get by simply searching for single index terms with a full-text search in the use case descriptions of existing projects. This full-text search also would not reveal the actual **distance relations between**

the new use cases and potential matches. By graphically visualizing these similarities in terms of distances on the map, users will gain a much better understanding in a short time. Moreover, they could use the added information provided by the **linking to past projects** in various ways.

(a) They might probably want to **reuse existing use case descriptions, domain descriptions, or domain models**;

(b) They might want to **review and eventually revise the new use case descriptions** to incorporate specific details or peculiarities pointed out in related descriptions;

(c) They will probably **contact the authors** of use cases that are very similar to the new ones to find out more about the domain or reuse possibilites, thus **uncovering tacit knowledge** present in the organization. This informal support of knowledge transfer would not be possible with traditional reuse approaches. Furthermore, the provided linkage to members of an organization who are knowledgeable in the domain of a software project may complement traditional knowledge management systems.

Further on, it could also be advisable to transfer responsibilities for complex domain-specific questions to **colleagues who are already knowledgeable** in a specific area and know its peculiarities.

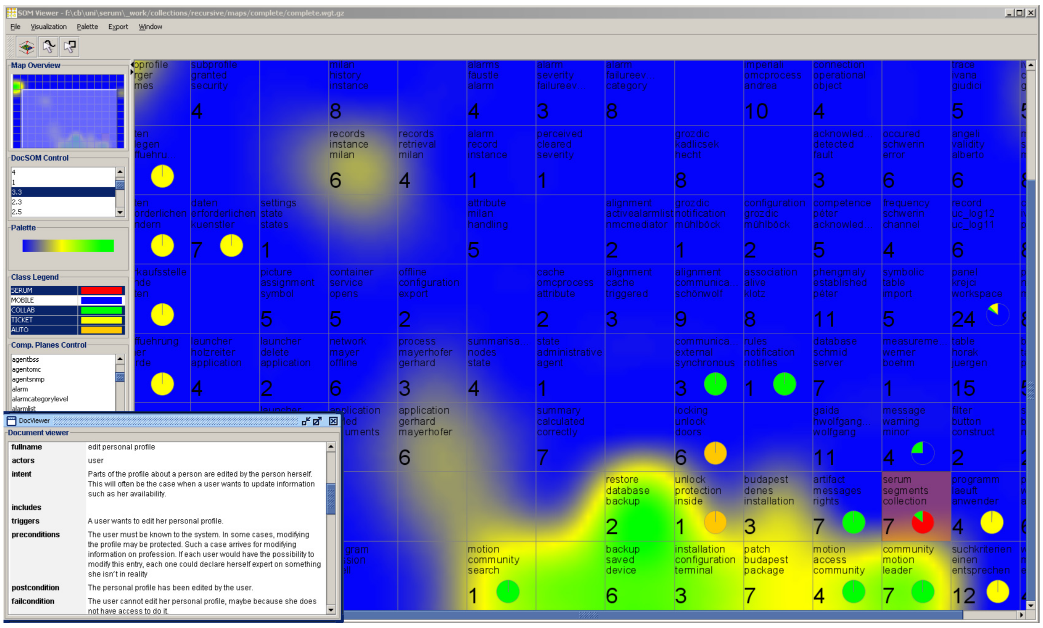
(d) Most importantly, users analyzing a use case map might probably **find reuse candidates** not only for use case descriptions, but also for **software artifacts implementing** parts of or even the whole **functionality** described in the new use cases.

2. They might also **find isolated use case descriptions** that cover new terrain in terms of target domains; this might be an indicator to exercise special caution.

However, for a thorough judgement about the practical applicability and to determine the benefits actually gained in practice, extensive experiments are needed; we will refer to that in the next chapter.

4.6 A recursive view of SERUM

In Section 3.3, we described the requirements for SERUM with use cases. We now use these documents together with all four use case sets described in Section 4.1 and put them together into one map.



fullname	edit personal profile
actors	user
intent	Parts of the profile about a person are edited by the person herself. This will often be the case when a user wants to update information such as her availability.
includes	
triggers	A user wants to edit her personal profile.
preconditions	The user must be known to the system. In some cases, modifying the profile may be protected. Such a case arrives for modifying information on profession. If each user would have the possibility to modify this entry, each one could declare herself expert on something she isn't in reality
postcondition	The personal profile has been edited by the user.
failcondition	The user cannot edit her personal profile, maybe because she does not have access to do it.

Figure 4.11: SOMViewer displaying a map with all use cases, including SERUM's own.

Figure 4.11 presents the result of this effort, a 15x10 static SOM of 569 use case documents with a total of about 1600 dimensions. Of course, because these documents come from five different domains and are of radically different complexity, length, and even language, the clusters correspond by and large to the project boundaries. But on the cluster boundaries, we can see that some descriptions do indeed overlap a bit, which is also reflected in the labels that were identified by the LabelSOM algorithm. On the screenshot provided in Figure 4.11, the component plane of the term *administrator* is visualized; the highlighted cell in the lower right contains all SERUM use cases. Their nearest neighbors are mostly use case descriptions from the COLLAB, MOBILE and TICKET data sets.

4.7 Performance

Performance values measuring the time required to perform the various steps from extraction to map training are provided in Table 4.6. Note that similar to the `IndexWriter` of TeseT, the `HtmlOutputter` of the SOM Toolbox was optimized and brought up to speed by a factor of 3.

These values were measured on a 32-bit 2,4GHz machine with 1GB RAM

Step	**TICKET**	**AUTO**	**COLLAB**	**MOBILE**
Read and split file (sec)	0.5	2.5	3	76
Extract use cases (sec)	1	1	1	38
Match a single candidate document to the pattern (minimum ms)	1	1	1	15
Match a single candidate document to the pattern (maximum ms)	16	15	94	562
Write segments to disk (sec)	2	1	1	31
Create index without stemmer (sec)	2	1	2	17
Create index with stemmer (sec)	2.5	1	2	17
Create and reduce index without stemmer (sec)	2.5	1	2	17
Create and reduce index with stemmer (sec)	3	1	2	17
Dimensions after reduction	277	74	112	1167
Train a static map (50000 iterations):				
Map size	10x10	6x4	6x4	15x10
Training time (sec)	15	10	10	300
Train a growing map (10000 iterations between expansion checks):				
Map size	7x5	4x4	4x5	8x5
Training time (sec)	20	7	8	98
Write html output: (ms)	47	46	32	328

Table 4.6: Time required for performing the various steps from document extraction to map training

running Windows XP. Logging was activated with loglevel DEBUG; it can be safely assumed that turning it off would considerably speed up map training and extraction because of the large amount of log messages appended both to the Chainsaw socket and written to the filesystem (e. g., about 20000 for a complete run from extracting to training the MOBILE dataset). τ was set to 0.01 in the Growing SOM starting with a size of 3x3; 4 labels were assigned to each unit. The stemmed and automatically reduced index was used as input for the map training.

The table shows that the usage of stemmers and automatic reduction does not have a measurable impact on the processing time. Moreover, even complex documents can be extracted and corresponding maps trained within reasonable waiting time for the user.

Chapter 5

Summary and Outlook

In this thesis, we described a tool enhancement extending the components of the existing SOMLib digital library system and related tools to form an application that we called *SERUM – SElf-oRganizing Use case Maps*.

The first two chapters introduced the concepts of digital libraries and self-organizing maps and explained the steps necessary to organize document collections based on their content by means of SOMs. Chapter 2 further introduced a type of document that is of particular interest for our research, namely use cases.

Self-organizing maps are a powerful tool for clustering high-dimensional input data and have been proven to be very useful in the areas of digital library systems and content-based organization of large document collections. The topology-preserving mapping to a 2-dimensional output space is particularly useful, because it allows the user to explore potentially unknown collections in an intuitive way and easily grasp the clusters and similarities of the contained documents.

If an organization uses a digital library approach to organize its collections of use cases with an unsupervised learning technology such as the self-organizing map, it would be able to create such a library without extensive investment, as no supervised classification schemes are employed.

Whenever a new project is undertaken and its requirements are specified with use cases, mapping these use case descriptions to the existing collection may reveal valuable similarities that might not be uncovered by traditional information retrieval methods such as keyword-based search.

This approach may be of interest to areas such as software reuse and knowledge management. In software reuse, the process of **finding suitable reuse candidates** for a given, specified problem is still one of the main problems to solve in each potential reuse instance.

Leading researchers in this field plead to rely on artifacts on the level of the problem space for the retrieval of reuse candidates – as opposed to the often dominating usage of artifacts in the solution space coming from later stages, such as source code documentation. Use cases as one of the leading methodologies in software requirements engineering thus are the perfect candidate artifacts, and the potential data base for experiments in this area is very large.

In knowledge management, a critical problem is pointing a user to the correct source of information for a specific problem, which often means referring him to a colleague holding valuable information in form of **tacit knowledge**. With use case descriptions pointing to their responsible authors, this can be achieved with minimum effort.

However, to cluster use cases in a self-organizing map, these use case descriptions need to be extracted from larger requirements documents, as most organizations do not rely on dedicated requirements management tools, but simply on text-processing applications, for specifying the requirements for their projects. This extraction process can be a very tediuos task, prohibitively time-consuming and expensive for many organizations; this also hinders further experiments in this area. What is needed thus is a tool that allows practitioners and researchers to extract document passages roughly following a common pattern from larger documents without substantial effort, and support the complete workflow through the successive steps of text indexing, feature space reduction and training the SOMs.

Such a tool, implemented in a generic way, would also be of great benefit to every similar application of self-organizing maps where we need to extract documents that contain segments following a common pattern. An obvious example are scientific publications consisting of a heading, an abstract, authors, keywords, several sections of texts, and a bibliography.

Moreover, it opens up further options of working only with selected sections of these texts, such as using only the abstracts or solely the bibliographies of publications for clustering, etc.

In Chapter 3 of this work, we proposed such a tool, outlined the requirements, and described the main building blocks and supporting components of SERUM. We described critical aspects as well as the user interface and the steps of the workflow.

Chapter 4 described the application of SERUM to four different collections of use cases with differing size, target domains, sources and languages. We highlighted selected aspects of the feature extraction and reduction steps, described the resulting maps and interaction features, and pointed out possible benefits to software reuse and knowledge management.

A small homepage of SERUM is currently (as of March 2006) located at `http://athena.ifs.tuwien.ac.at/~becker`. The interested reader will find there a short introduction to the tool, as well as a deployment using Java Webstart that enables him to instantly start the application.

5.1 Outlook

While the current status of SERUM is already quite productive, a lot of opportunities for follow-up research and extension options for the tool have opened.

- To support software reuse, it might be necessary to extend the existing tool with a **domain model** describing software projects, their artifacts, attributes and relations, thus **semantically enriching use case collections**.
- This may be accompanied by more **sophisticated feature extraction modules**, for example by incorporating segments that represent numbers, dates, currencies, etc., or by additionally relying on *structural markup* information for feature extraction.
- To examine the practical applicability of the outlined approach to software reuse, detailed **experiments**, preferably with industrial partners and data sets from practice, have to be carried out.
- The effects of different feature reduction settings regarding word frequency thresholds, word stemming, etc. on the resulting maps, in terms of the MQEs of maps and labellings and other metrics, should be explored.
- Similarly, the **effects of using only specific clusters for self-organization**, for example only the condition sections of use case descriptions, on the resulting maps are yet to be uncovered.
- A related question is how to **handle documents that vary radically in length** – e. g., inconsistent collections consisting both of use case briefs and fully dressed use cases, or collections containing paper abstracts as well as full papers.
- **Feature extraction modules** operating on different input types such as non-textual input data could be integrated into the tool.

These and further questions will be addressed in future research efforts, for which this thesis provides the basis.

Bibliography

[ABRB05] Martin Auer, Christoph Becker, Andreas Rauber, and Stefan Biffl. Implicit analogy-based cost estimation using textual use case similarities. In *Proceedings of the Second International Conference on Intelligent Computing and Information Systems (ICICIS'05)*, pages 369–376, Cairo, March 2005. ACM.

[AHS98] D. Alahakoon, S.K. Halgamuge, and B. Srinivasan. A self-growing cluster development approach to data mining. In *1998 IEEE International Conference on Systems, Man, and Cybernetics*, pages 2901–2906, October 1998.

[AHS00] D. Alahakoon, S. K. Halgamuge, and B. Srinivasan. Dynamic self-organizing maps with controlled growth for knowledge discovery. *IEEE Transactions on Neural Networks*, 11(3):601–614, May 2000.

[Ass06] Assocation for Computing Machinery (ACM). ACM Digital Library. Website, May 2006. `http://www.acm.org/dl`, as of February 2006.

[BB03a] James Brittle and C. Boldyreff. Genisom: Self-organizing maps applied in visualising large software collections. In *Proceedings of the 2nd International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT)*, pages 60–61, Amsterdam, Netherlands, 2003.

[BB03b] James Brittle and Cornelia Boldyreff. Self-organizing maps applied in visualising large software collections. In *Proceedings of the IEEE VISSOFT 2003*, 2003.

[BCM+92] Victor Basili, Gianluigi Caldiera, Frank McGarry, Rose Pajerski, Gerald Page, and Sharon Waligora. The software engineering laboratory: an operational software experience factory. In *ICSE*

'92: Proceedings of the 14th international conference on Software engineering, pages 370–381, New York, NY, USA, 1992. ACM Press.

[BFN04] José Borbinha, Nuno Freire, and João Neves. Bnd: The architecture of a national digital library. In *Proceedings of the 2004 Joint ACM/IEEE Conference on Digital Libraries (JCDL'04)*, pages 21–22. ACM, 2004.

[BM93] J. Blackmore and R. Miikkulainen. Incremental grid growing: Encoding high-dimensional structure into a two-dimensional feature map. In *Proceedings of the IEEE International Conference on Neural Networks (ICNN'93)*, volume 1, pages 450–455, San Francisco, CA, USA, 1993. `http://ieeexplore.ieee.org/`.

[BV97] H.-U. Bauer and T. Villmann. Growing a hypercubical output space in a self-organizing feature map. *IEEE Transactions on Neural Networks*, 8(2):226 – 233, 1997.

[CBC+99] H.B. Chen, O. Bimber, C. Chhatre, E. Poole, and S. J. Buckley. eSCA: a thin-client/server/web-enabled system for distributed supply chain simulation. In *Proceedings of the 1999 Winter Simulation Conference*, 1999.

[Clo06] Cloudgarden. Jigloo SWT/Swing GUI Builder for Eclipse and Websphere. Website, February 2006. `http://www.cloudgarden.com/jigloo/`.

[Coc00] Alistair Cockburn. *Writing Effective Use Cases*. The Agile Software Development Series. Addison-Wesley, 2000.

[CT94] William B. Cavnar and John M. Trenkle. N-gram-based text categorization. In *Proceedings of SDAIR-94, 3rd Annual Symposium on Document Analysis and Information Retrieval*, pages 161–175, Las Vegas, US, 1994.

[Dit00] Michael Dittenbach. The growing hierarchical self-organizing map: Uncovering hierarchical structure in data. Master's thesis, Technische Universität Wien, 2000.

[DMR00] M. Dittenbach, D. Merkl, and A. Rauber. The growing hierarchical self-organizing map. In S. Amari, C. L. Giles, M. Gori, and V. Puri, editors, *Proceedings of the International Joint Conference on Neural Networks (IJCNN 2000)*, volume VI, pages 15 – 19, Como, Italy, July 24-27 2000. IEEE Computer Society. `http://www.ifs.tuwien.ac.at/ifs/research/publications.html`.

[Dor04] H. D. Doran. Agile knowledge management in practice. In *Advances in Learning Software Organizations: 6th Workshop, LSO'2004*, 2004.

[Fou05] The Apache Software Foundation. Log4j Project – Introduction. Website, December 2005. `http://logging.apache.org/log4j/docs/index.html`.

[Fou06a] The Apache Software Foundation. Apache Lucene – Overview. Website, February 2006. `http://lucene.apache.org/java/docs/index.html`.

[Fou06b] The Apache Software Foundation. Chainsaw v2 Documentation. Website, February 2006. `http://logging.apache.org/log4j/docs/chainsaw.html`.

[Fou06c] The Eclipse Foundation. Swt: The standard widget toolkit. Website, February 2006. `http://www.eclipse.org/swt/`.

[Fre83] P. Freeman. Reusable software engineering: Concepts and research directions. In Alan Perlis, editor, *Proc. Workshop on Reusability in Programming*, pages 2–16, Newport, 1983. ITT Programming.

[Fri91] Bernd Fritzke. Let it grow – self-organizing feature maps with problem dependent cell structure. In *Proceedings of the 1991 International Conference on Artificial Neural Networks (ICANN'91)*, 1991.

[Fri94] B. Fritzke. Growing cell structures – A self-organizing network for unsupervised and supervised learning. *Neural Networks*, 7(9):1441 – 1460, 1994. `http://pikas.inf.tu-dresden.de/~fritzke`.

[Fri95] B. Fritzke. Growing Grid – A self-organizing network with constant neighborhood range and adaption strength. *Neural Processing Letters*, 2(5):1 – 5, 1995. `http://pikas.inf.tu-dresden.de/~fritzke`.

[GL00] Jiang Guo and Luqi. A survey of software reuse repositories. In *Proceedings of the 7th IEEE International Conference and Workshop on the Engineering of Computer Based Systems*, pages 92–100, Edinburgh, UK, April 2000. IEEE.

[Gun03] Jeff Gunther. Deploy an SWT application using Java Web Start. *IBM developerWorks*, June 2003. `http://www-128.ibm.com/developerworks/opensource/library/os-jws/`.

[Har96] S. P. Harter. What is a digital library? Definitions, content, and issues. In *Proceedings of the International Conference on Digital Libraries and Information Services for the 21st Century (KOLISS DL96)*, Seoul, Korea, September 1996. `http://php.indiana.edu/~harter/korea-paper.htm`.

[HM04] Harald Holz and Grigori Melnik. Research on learning software organizations – past, present, and future. In *Advances in Learning Software Organizations: 6th Workshop, LSO'2004*, pages 1–6, 2004.

[Hor] Joseph A. Horvath. Working with tacit knowledge. Whitepaper, IBM Institute for Knowledge Management, 2000.

[Hor00] Joseph A. Horvath. *The Knowledge Management Yearbook 2000-2001*, chapter Working with tacit knowledge. Elsevier, 2000.

[iee06] IEEE xplore. Website, February 2006. `http://ieeexplore.ieee.org/`.

[Int] Internet Archive. Website. `http://www.archive.org`.

[JCJv93] I. Jacobson, M. Christerson, P. Jonsson, and G. Övergaard. *Object-oriented software engineering: a use-case driven approach.* Addison-Wesley, 1993.

[JEJ94] I. Jacobson, M. Ericsson, and A. Jacobson. *The Object Advantage.* ACM Press Books, 1994.

[JGJ97] Ivar Jacobson, Martin Griss, and Patrik Jonsson. *Software Reuse: Architecture, Process and Organization for Business Success.* ACM Press, 1997.

[JHC99] Cunny Johansson, Patrik Hall, and Michael Coquard. Talk to Paula and Peter – they are experienced. the experience engine in a nutshell. In *Learning Software Organizations: Methodology and Applications. 11th International Conference on Software Engineering and Knowledge Engineering, SEKE'99.*, number 1756 in Lecture Notes in Computer Science, pages 171–185. Springer, 1999.

[KKK98] S. Kaski, J. Kangas, and T. Kohonen. Bibliography of self-organizing map (SOM) papers 1981-1997. *Neural Computing Surveys*, 1(3&4):1–176, 1998. `http://www.icsi.berkeley.edu/~jagota/NCS/vol1.html`.

[KKL+00] T. Kohonen, S. Kaski, K. Lagus, J. Salojärvi, J. Honkela, V. Paatero, and A. Saarela. Self-organization of a massive document collection. *IEEE Transactions on Neural Networks*, 11(3):574–585, May 2000.

[KKLH96] Teuvo Kohonen, Samuel Kaski, Krista Lagus, and Timo Honkela. Very large two-level SOM for the browsing of newsgroups. In *Proceedings of the International Conference on Artificial Neural Networks (ICANN96)*, Lecture Notes in Computer Science, vol. 1112, pages 269–274. Springer, Bochum, Germany, July, 16 - 19 1996.

[KO90] P. Koikkalainen and E. Oja. Self-organizing hierarchical feature maps. In *Proceedings of the International Joint Conference on Neural Networks*, volume 2, pages 279 – 284, San Diego, CA, 1990.

[Koh82] T. Kohonen. Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, 43:59–69, 1982.

[Koh89] T. Kohonen. *Self-Organization and Associative Memory.* Springer Verlag, Berlin, Germany, 3 edition, 1989.

[Koh01] T. Kohonen. *Self-organizing maps.* Springer-Verlag, Berlin, 3rd edition, 2001.

[Kra92] M.A. Kraaijveld. A non-linear projection method based on kohonen's topology preserving maps. In *Proceedings of the 11th IAPR International Conference on Pattern Recognition, Vol.II. Conference B: Pattern Recognition Methodology and Systems*, volume II, pages 41–45, 1992.

[Lag00] Krista Lagus. *Text Mining with the WEBSOM.* PhD thesis, Helsinki University of Technology Neural Networks Research Centre, 2000.

[Les97] M. Lesk. *Practical Digital Libraries: Books, Bytes, and Bucks.* Morgan Kaufmann, San Francisco, CA, 1997.

[LGR04] Alberto H. F. Laender, Marcos André Gonçalves, and Pablo A. Roberto. BDBComp: building a digital library for the Brazilian computer science community. In *Proceedings of the 2004 Joint ACM/IEEE Conference on Digital Libraries (JCDL'04)*, pages 23–24. ACM, 2004.

[Lin91] X. Lin. A self-organizing semantic map for information retrieval. In *Proceedings of the 14th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR91)*, pages 262–269, Chicago, IL, October 13 - 16 1991. ACM. `http://www.acm.org/dl`.

[LKK04] Krista Lagus, Samuel Kaski, and Teuvo Kohonen. Mining massive document collections by the websom method. *Information Sciences*, 163(1–3):135–136, 2004.

[LM95] David M. Levy and Catherine C. Marshall. Going digital: a look at assumptions underlying digital libraries. *Communications of the ACM*, 38(4):77–84, 1995.

[McI69] M. D. McIlroy. Mass-produced software components. In J. M. Buxton, P. Naur, and B. Randell, editors, *Software Engineering Concepts and Techniques: Proceedings of the 1968 NATO Conference on Software Engineering*, New York, 1969. Petrocelli/Charter.

[MET02] Maurizio Morisio, Michel Ezran, and Colin Tully. Success and failure factors in software reuse. *IEEE Transactions on Software Engineering*, 28(4):340–357, 2002.

[Mii90] R. Miikkulainen. Script recognition with hierarchical feature maps. *Connection Science*, 2:83 – 101, 1990.

[MMYA01] Hafedh Mili, Ali Mili, Sherif Yacoub, and Edward Addy. *Reuse based software engineering : techniques, organization, and measurement.* Wiley, December 2001.

[MPP+94] F. McGarry, R. Pajerski, G. Page, S. Waligora, V.R. Basili, and M.V.Zelkovitz. Software process improvement in the NASA software engineering laboratory. Technical Report CMU/SEI-94-TR-22, Software Engineering Institute, 1994.

[MR99] D. Merkl and A. Rauber. Automatic labeling of self-organizing maps for information retrieval. In *Proceedings of the 6th International Conference on Neural Information Processing (ICONIP99)*, Perth, Australia, November 16 - 20 1999. `http://www.ifs.tuwien.ac.at/ifs/research/publications.html`.

[MTK94] Dieter Merkl, A Min Tjoa, and Gerti Kappel. A self-organizing map that learns the semantic similarity of reusable software components. In *Proceedings of the 5th Australian Conference on Neural Networks (ACNN'94)*, pages 13–16, Brisbane, Australia, Jan 31 - Feb 2 1994.

[NKK03] Andreas Nürnberger, Aljoscha Klose, and Rudolf Kruse. *Intelligent Exploration of the Web*, chapter Self-Organizing Maps for Interactive Search in Document Databases, pages 119–135. Studies in Fuzziness and Soft Computing. Physica Verlag Heidelberg New York, 2003.

[NMB02] E. Nasr, L. McDermid, and G. Bernat. Eliciting and specifying requirements with use cases for embedded systems. In *Proceedings of the 7th International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS'02)*, pages 350–357, January 2002.

[NSF06] Library of Congress et.al. National Science Foundation. Digital libraries initiative phase 2. Website, February 2006. `http://www.dli2.nsf.gov/`.

[OP04] Gunnar Övergaard and Karin Palmkvist. *Use Cases : Patterns and Blueprints.* The Software Patterns Series. Addison-Wesley Professional, November 2004.

[Ope06] Open Source Technology Group. Sourceforge. Website, February 2006. `http://sourceforge.net`.

[PD91] Rubén Prieto-Díaz. Implementing faceted classification for software reuse. *Communications of the ACM*, 34(5):88–97, May 1991.

[PD93] R. Prieto-Diaz. Status report: software reusability. *IEEE Software*, 10(3):61–66, May 1993.

[Por80] M. F. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, July 1980. `http://open.muscat.com/developer/docs/porterstem.html`.

[PRM02] E. Pampalk, A. Rauber, and D. Merkl. Using smoothed data histograms for cluster visualization in self-organizing maps. In *Proceedings of the International Conference on Neural Networks (ICANN 2002)*, pages 871–876, Madrid, Spain, August 27-30 2002. Springer.

[Pro06a] The Apache Jakarta Project. Commons Digester. Website, February 2006. `http://jakarta.apache.org/commons/digester/`.

[Pro06b] The Apache Jakarta Project. Jakarta Commons. Website, February 2006. `http://jakarta.apache.org/commons/`http://jakarta.apache.org/commons/logging/.

[Pro06c] The Apache Jakarta Project. Jakarta POI - Java API to access Microsoft format files. Website, February 2006. `http://jakarta.apache.org/poi/`.

[Pro06d] The PDFBox Project. PDFBox - Java PDF Library. Website, February 2006. `http://www.pdfbox.org/`.

[PY93] J. S. Poulin and K.P. Yglesias. Experiences with a faceted classification scheme in a large reusable software library (RSL). In *Proceedings of the Seventeenth Annual International Computer Software and Applications Conference, 1993 (COMPSAC 93)*, pages 90–99, Phoenix,AZ, November 1993. IEEE.

[Rau98] A. Rauber. SOMLib: A distributed digital library system based on self-organizing maps. In M. Marinaro and R. Tagliaferri, editors, *Proceedings of the 10th Italian Workshop on Neural Nets (WIRN98)*, Perspectives in Neural Computing, pages 365–370, Vietri sul Mare, Italy, May 21 - 23 1998. Springer. `http://www.ifs.tuwien.ac.at/ifs/research/publications.html`.

[Rau99] A. Rauber. LabelSOM: On the labeling of self-organizing maps. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN'99)*, Washington, DC, July 10

- 16 1999. http://www.ifs.tuwien.ac.at/ifs/research/publications.html.

[Rau00] A. Rauber. *Digital Libraries or: The Art of Storing, Drawing, and Exploring Document Collections.* PhD thesis, Vienna University of Technology, Favoritenstr. 9-11, A - 1040 Vienna, Austria, November, 11 2000. http://www.ifs.tuwien.ac.at/ifs/research/publications.html.

[Rau03] A. Rauber. SOMLib - New approaches for information presentation and handling. *ERCIM News*, 52:45–46, January 2003. http://www.ercim.org/publication/Ercim_News/enw52/rauber.html.

[RB99] A. Rauber and H. Bina. A metaphor graphics based representation of digital libraries on the World Wide Web: Using the libViewer to make metadata visible. In A.M. Tjoa, A. Cammelli, and R.R. Wagner, editors, *Proceedings of the DEXA-Workshop on Web-based Information Visualization (WebVis99)*, pages 286–290, Florence, Italy, September 1 - 3 1999. IEEE. http://www.ifs.tuwien.ac.at/ifs/research/publications.html.

[RB00a] A. Rauber and H. Bina. Visualizing electronic document repositories: Drawing books and papers in a digital library. In H. Arisawa and T. Catarci, editors, *Advances in Visual Database Systems: Proceedings of the IFIP TC2 WG2.6 5th Working Conference on Visual Database Systems*, pages 95 – 114, Fukuoka, Japan, May, 10 - 12 2000. Kluwer Academic Publishers. http://www.ifs.tuwien.ac.at/ifs/research/publications/.

[RB00b] Günter Ruhe and Frank Bomarius, editors. *Learning Software Organizations: Methodology and Applications. Proceedings of the 11th International Conference on Software Engineering and Knowledge Engineering, SEKE'99*, volume 1756 of *Lecture Notes in Computer Science (LNCS).* Springer Verlag, 2000.

[RM98] A. Rauber and D. Merkl. Creating an order in distributed digital libraries by integrating independent self-organizing maps. In L. Niklasson, M. Boden, and T. Ziemke, editors, *Proceedings of the International Conference on Artificial Neural Networks (ICANN'98)*, Perspectives in Neural Computing, Skövde, Sweden, September 2 - 4 1998. Springer. http://www.ifs.tuwien.ac.at/ifs/research/publications.html.

[RM99a] A. Rauber and D. Merkl. Automatic labeling of self-organizing maps: Making a treasure map reveal its secrets. In N. Zhong and L. Zhou, editors, *Proceedings of the 3rd Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD99)*, number LNCS/LNAI 1574 in Lecture Notes in Artificial Intelligence, pages 228–237, Beijing, China, April 26 - 29 1999. Springer. `http://www.ifs.tuwien.ac.at/ifs/research/publications.html`.

[RM99b] A. Rauber and D. Merkl. Mining text archives: Creating readable maps to structure and describe document collections. In J.M. Zytkow and J. Rauch, editors, *Proceedings of the 3rd European Conference on Principles of Data Mining and Knowledge Discovery*, number LNCS/LNAI 1704 in Lecture Notes in Artificial Intelligence, pages 524–529, Prague, Czech Republic, September 15 - 18 1999. Springer. `http://www.ifs.tuwien.ac.at/ifs/research/publications.html`.

[RM99c] A. Rauber and D. Merkl. SOMLib: A digital library system based on neural networks. In E.A. Fox and N. Rowe, editors, *Proceedings of the ACM Conference on Digital Libraries (ACMDL'99)*, pages 240–241, Berkeley, CA, August 11 - 14 1999. ACM. `http://www.ifs.tuwien.ac.at/ifs/research/publications.html`.

[RM99d] A. Rauber and D. Merkl. The SOMLib Digital Library System. In S. Abiteboul and A.M. Vercoustre, editors, *Proceedings of the 3rd European Conference on Research and Advanced Technology for Digital Libraries (ECDL99)*, number LNCS 1696 in Lecture Notes in Computer Science, pages 323–342, Paris, September 1999. Springer.

[RM99e] A. Rauber and D. Merkl. Using self-organizing maps to organize document collections and to characterize subject matters: How to make a map tell the news of the world. In T. Bench-Capon, G. Soda, and A.M. Tjoa, editors, *Proceedings of the 10th International Conference on Database and Expert Systems Applications (DEXA99)*, number LNCS 1677 in Lecture Notes in Computer Science, pages 302–311, Florence, Italy, September 1 - 3 1999. Springer. `http://www.ifs.tuwien.ac.at/ifs/research/publications.html`.

[RM03] A. Rauber and D. Merkl. Text mining in the SOMLib digital library system: The representation of topics and genres. *Applied Intelligence*, 18(3):271–293, May-June 2003. http://www.kluweronline.com/issn/0924-669X/current.

[RMD02] A. Rauber, D. Merkl, and M. Dittenbach. The growing hierarchical self-organizing map: Exploratory analysis of high-dimensional data. *IEEE Transactions on Neural Networks*, 13(6):1331–1341, November 2002.

[SAB94] Gerard Salton, James Allan, and Chris Buckley. Automatic structuring and retrieval of large text files. *Communications of the ACM*, 37(2):97–108, February 1994.

[Sal75] Gerard Salton. A theory of indexing. In *Regional conference series in applied mathematics*, volume 18. Society for Industrial and Applied Mathematics, 1975.

[SBJ04] Brian Sam-Bodden and Christopher Judd. Rich clients with the SWT and JFace. *JavaWorld*, April 2004.

[SJP02] F.T. Sheldon, K. Jerath, and O. Pilskalns. Case study: B2B e-commerce system specification and implementation employing use-case diagrams, digital signatures and XML. In *Proceedings of the 4th International Symposium on Multimedia Software Engineering (MSE'02)*, pages 106–113, December 2002.

[Sol06] Knallgrau New Media Solutions. Java text categorizing library. Website, February 2006. `http://textcat.sourceforge.net/`.

[SWY75] G. Salton, A. Wong, and C. S. Yang. A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620, November 1975.

[SY73] G. Salton and C.S. Yang. On the specification of term values in automatic indexing. *Journal of Documentation*, 29(4):351–372, 1973.

[Tal06] Joe Tallet. Gof design patterns. Website, February 2006. `http://www.acm.org/sigada/wg/patterns/patterns/GOF_Toc.html`.

[TS04] Songsri Tangsripairoj and M. H. Samadzadeh. Application of self-organizing maps to software repositories in reuse-based software development. In Hamid R. Arabnia and Hassan Reza, editors, *Proceedings of the 2004 International Conference on Software Engineering Research and Practice (SERP'04)*, volume II, pages 741–747, Las Vegas, Nevada, June 2004.

[TS05] Songsri Tangsripairoj and M. H. Samadzadeh. Organizing and visualizing software repositories using the growing hierarchical self-organizing map. In *Proceedings of the 20th ACM Symposium on Applied Computing (SAC'05), Software Engineering Track*, pages 1539–1545, Santa Fe, New Mexico, March 2005.

[US89] A. Ultsch and H. Siemon. Exploratory data analysis: Using kohonen's topology preserving maps. Technical Report 329, University of Dortmund, 1989.

[VZM+97] V.R.Basili, M.V. Zelkovitz, F. McGarry, J. Page, S. Waligora, and R. Pajerski. SEL's software process improvement program. *IEEE Software*, 12(6):83–87, 1997.

[YL01] H. Ye and B.W.N. Lo. Towards a self-structuring software library. In *IEE Proceedings-Software*, volume 148, pages 45–55, April 2001.

Appendix A

Document type configuration file

The following listing displays the complete configuration file `doctypes.xml` used for the four use case sets discussed in Chapter 4.

```
<?xml version="1.0" encoding="UTF-8"?> <!DOCTYPE doctypes [

    <!ELEMENT doctypes (doctype+)>
    <!ELEMENT doctype (textpattern)>
    <!ELEMENT textpattern(removelist,delimiter,(segment|line)+)>
    <!ELEMENT removelist(remove+)>
    <!ELEMENT delimiter(#PCDATA)>
    <!ELEMENT segment (prefix)>
    <!ELEMENT line (prefix)>
    <!ELEMENT prefix (#PCDATA)>

    <!ATTLIST doctype
        name CDATA #REQUIRED
        type CDATA #REQUIRED
        directory (true|false) "false"
    >

    <!ATTLIST segment
        name CDATA #REQUIRED
        id (true|false) "false"
        title (true|false) "false"
        optional (true|false) "false"
    >

    <!ATTLIST line
        name CDATA #REQUIRED
        id (true|false) "false"
        title (true|false) "false"
        optional (true|false) "false"
    >

    <!ATTLIST prefix
        line (true|false) "false"
        startline (true|false) "false"
        maystartline (true|false) "false"
    >
]>

<doctypes>
    <doctype name="simple textfiles in directory" type="simple" directory="true">
        <textpattern>
            <removelist>
            </removelist>
             <delimiter>
             </delimiter>
             <segment name="content">
                <prefix></prefix>
            </segment>

        </textpattern>
    </doctype>

    <doctype name="simple text in document" type="simple">
        <textpattern>
            <removelist>
            </removelist>
             <delimiter>
                \n\n
                <!-- just an idea, could be any other -->
             </delimiter>
             <segment name="content">
                <prefix></prefix>
            </segment>

        </textpattern>
    </doctype>

    <doctype name="MOBILE" type="usecase">
        <textpattern>
            <removelist>
                <remove>\s\d\d?-\d\d?\d?\s</remove>
            </removelist>
             <delimiter>((USE CASE)|(Use Case)) DESCRIPTION \r?\n?</delimiter>
            <!--  sequence of segment patterns -->
            <line name="Goal in Context" title="true" optional="true">
              <prefix>Goal in Context</prefix>
            </line>
            <line name="id" id="true">
                <prefix>Use\s+Case\s+UC_</prefix>
            </line>
            <segment name="Covered feature requests" optional="true">
              <prefix startline="true">Covered\s+feature\s+</prefix>
            </segment>
            <line name="Goal in Context" title="true" optional="true">
              <prefix maystartline="true">Goal in Context</prefix>
            </line>
            <segment name="Creator-Responsible">
                <prefix line="true">Creator(s)?\s+(Responsible\s)?</prefix>
            </segment>
            <segment name="History of changes">
                <prefix line="true">History of\s+(Changes\s+)?</prefix>
            </segment>
            <line name="Scope">
                <prefix startline="true">Scope</prefix>
            </line>
            <segment name="Preconditions">
                <prefix startline="true">Preconditions</prefix>
            </segment>
```

```
<segment name="Sucess end conditions">
    <prefix line="true">Success End\s+Condition\s+</prefix>
</segment>
<segment name="Failed end conditions">
    <prefix line="true">Failed End\s+Condition\s+</prefix>
</segment>

<segment name="Primary actor">
    <prefix startline="true">Primary Actor</prefix>
</segment>

<segment name="Secondary actors">
    <prefix line="true">Secondary\s+Actor\(s\)\s</prefix>
</segment>

<segment name="Trigger">
    <prefix startline="true">Trigger</prefix>
</segment>

<segment name="Scenario sheet">
    <prefix line="true">\sSCENARIO SHEET\s+(Use Case\s+Scenario\s+)?Step
    \s+Action\s?</prefix>
</segment>

<segment name="Scenario extensions" optional="true">
    <prefix maystartline="true">Extens(t)?ion to\s+Scenario\s+Steps\s+Step
    (\s+Branching\s+Action)?\s{0,2}</prefix>
</segment>

<segment name="Scenario variations" optional="true">
    <prefix maystartline="true">Variation(s)? to\s+Scenario\s+Steps\s+Step
    (\s+Branching\s+Action)?\s{0,2}</prefix>
</segment>
<segment name="Scenario extensions" optional="true">
    <prefix maystartline="true">Extens(t)?ion to\s+Scenario\s+Steps\s+Step
    (\s+Branching\s+Action)?\s{0,2}</prefix>
</segment>

<segment name="Superordinates" optional="true">
    <prefix startline="true">Superordinates</prefix>
</segment>

<segment name="Subordinates" optional="true">
    <prefix startline="true">Subordinates</prefix>
</segment>

<segment name="Quality issues - priority">
    <prefix startline="true">\s+QUALITY ISSUES\s+Priority:?\s?</prefix>
</segment>

<segment name="Time constraints">
    <prefix startline="true">Time Constrain(ts)?</prefix>
</segment>

<segment name="Frequency">
    <prefix startline="true">Frequency</prefix>
</segment>
<segment name="Channels to actors">
    <prefix startline="true">((Channels)|(connections)) to\s+(actors)?</prefix>
</segment>
<segment name="Open issues">
    <prefix startline="true">OPEN\s+ISSUES</prefix>
```

```
        </segment>

        <segment name="Due date">
            <prefix startline="true">Due Date</prefix>
        </segment>

        <segment name="Other management information">
            <prefix line="true">...any other\s+(management\s+(i|I)nformation...)?\s?
            </prefix>
        </segment>
    </textpattern> <!-- mobnet -->
</doctype>
<doctype name="AUTO" type="usecase">
    <textpattern>
        <removelist>
            <remove>Seite \d* von \d*</remove>
        </removelist>
         <delimiter>UseCase </delimiter>
        <!-- sequence of segment patterns -->
        <line name="fullname" title="true" id="true">
          <prefix startline="false"></prefix>
        </line>
        <line name="actors">
            <prefix startline="true">Actors?.</prefix>
        </line>
        <segment name="intent">
            <prefix startline="true">Intent.</prefix>
        </segment>
        <segment name="preconditions">
            <prefix startline="true">Preconditions?.</prefix>
        </segment>
        <segment name="eventflow">
            <prefix startline="true">((Flow of events)|(Description)).</prefix>
        </segment>

        <segment name="exceptions">
            <prefix startline="true">Exceptions.</prefix>
        </segment>
        <segment name="rules">
            <prefix startline="true">Rules.</prefix>
        </segment>

        <segment name="qualityConstraints">
            <prefix startline="true">Quality constraints.</prefix>
        </segment>

        <segment name="monitor">
            <prefix startline="true">Monitored.{0,3}environ.{0,3}ment(al)?.{0,3}variables
            .?</prefix>
        </segment>

        <segment name="control">
            <prefix startline="true">Controlled.{0,3}environ.{0,3}ment(al)?.{0,3}variables
            .?</prefix>
        </segment>

        <segment name="postcondition">
            <prefix startline="true">Post.?conditions?.</prefix>
        </segment>

    </textpattern>
</doctype>
```

```
    <doctype name="COLLAB-mergeAUTO" type="usecase">
    <textpattern>
        <removelist>
            <remove>Revision: \d\.\d</remove>
            <remove>Date:  \d*/\d*\d*</remove>
            <remove>Page: \d* of \d*</remove>
        </removelist>
         <delimiter>Name:</delimiter>
        <!--  sequence of segment patterns -->
        <segment name="id" id="true">
            <prefix>U</prefix>
             <body>\d+\.\d+</body>
        </segment>
        <line name="fullname" title="true">
          <prefix startline="false">\s</prefix>
        </line>
        <line name="actors">
            <prefix line="true">Actors:</prefix>
        </line>
        <segment name="intent">
            <prefix line="true">Goal:</prefix>
        </segment>
        <segment name="includes" optional="true">
            <prefix line="true">Includes:</prefix>
        </segment>
        <segment name="triggers">
            <prefix line="true">Triggers:</prefix>
        </segment>
        <segment name="preconditions" optional="true">
            <prefix line="true">Preconditions:</prefix>
        </segment>
        <segment name="postcondition">
            <prefix line="true">Success end conditions:</prefix>
        </segment>
        <segment name="failcondition">
            <prefix line="true">Failed end conditions:</prefix>
        </segment>

    </textpattern>
</doctype>


<doctype name="COLLAB" type="usecase">
    <textpattern>
        <removelist>
            <remove>Revision: \d\.\d</remove>
            <remove>Date:  \d*/\d*\d*</remove>
            <remove>Page: \d* of \d*</remove>
        </removelist>
         <delimiter>Name:</delimiter>
        <!--  sequence of segment patterns -->
        <segment name="id" id="true">
            <prefix>U</prefix>
             <body>\d+\.\d+</body>
        </segment>
        <line name="name" title="true">
          <prefix startline="false">\s</prefix>
        </line>
        <line name="actors">
            <prefix line="true">Actors:</prefix>
        </line>
```

```
            <segment name="goal">
                <prefix line="true">Goal:</prefix>
            </segment>
            <segment name="includes" optional="true">
                <prefix line="true">Includes:</prefix>
            </segment>
            <segment name="triggers">
                <prefix line="true">Triggers:</prefix>
            </segment>
            <segment name="precondition" optional="true">
                <prefix line="true">Preconditions:</prefix>
            </segment>
            <segment name="sucesscondition">
                <prefix line="true">Success end conditions:</prefix>
            </segment>
            <segment name="failcondition">
                <prefix line="true">Failed end conditions:</prefix>
            </segment>

        </textpattern>
    </doctype>
    <doctype name="TICKET" type="usecase" directory="true">
        <!--  if directory=true, the field "id" of each text is set to the filename,
              and no delimiter is needed: each file in the directory contains one text
         -->
        <textpattern>
            <removelist>
                <!--  empty -->
            </removelist>
             <delimiter><!--  in directory obsolete --></delimiter>
            <!--  sequence of segment patterns -->
            <line name="name" title="true" id="true">
              <prefix startline="true">Titel:\s</prefix>
            </line>
            <line name="summary">
                <prefix startline="true">Kurzbeschreibung:\s</prefix>
            </line>
            <line name="precondition">
                <prefix startline="true">Vorbedingung(en)?:\s</prefix>
            </line>
            <segment name="eventflow">
                <prefix line="true">Beschreibung des Ablaufe?s:\s</prefix>
            </segment>
            <segment name="successcondition">
                <prefix line="true">Auswirkungen: </prefix>
            </segment>
            <segment name="nonfunctional">
                <prefix line="true">Nichtfunktionale Anforderung(en)?:\s</prefix>
            </segment>
            <segment name="notes">
                <prefix line="true">Anmerkungen:\s</prefix>
            </segment>
        </textpattern>
    </doctype>
    <doctype name="TICKET-NEU" type="usecase" directory="true">
        <!--  if directory=true, the field "id" of each text is set to the filename,
              and no delimiter is needed: each file in the directory contains one text
         -->
        <textpattern>
            <removelist>
                <!--  empty -->
            </removelist>
```

```
             <delimiter><!--  in directory obsolete --></delimiter>
            <!--  sequence of segment patterns -->
            <line name="Titel" title="true" id="true">
              <prefix startline="true">Titel:\s</prefix>
            </line>
            <line name="Kurzbeschreibung">
                <prefix startline="true">Kurzbeschreibung:\s</prefix>
            </line>
            <line name="Vorbedingungen">
                <prefix startline="true">Vorbedingung(en)?:\s</prefix>
            </line>
            <segment name="Beschreibung des Ablaufes">
                <prefix line="true">Beschreibung des Ablaufe?s:\s</prefix>
            </segment>
            <segment name="Auswirkungen">
                <prefix line="true">Auswirkungen: </prefix>
            </segment>
            <segment name="Nichtfunktionale Anforderungen">
                <prefix line="true">Nichtfunktionale Anforderung(en)?:\s</prefix>
            </segment>
            <segment name="Anmerkungen">
                <prefix line="true">Anmerkungen:\s</prefix>
            </segment>
        </textpattern>
    </doctype>

</doctypes>
```

Appendix B

Sample batch worker input file

The following listing displays a complete sample that serves as input for the batch processing module of SERUM.

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE serum-jobs [
    <!ELEMENT serum-jobs (collection+)>
    <!ELEMENT collection (imports,maps)>
    <!ELEMENT imports (import+)>
    <!ELEMENT import EMPTY>
    <!ELEMENT maps (map+)>
    <!ELEMENT map (segments)>
    <!ELEMENT segments (segment+)>
    <!ELEMENT segment #PCDATA>

    <!ATTLIST collection name CDATA #REQUIRED>
    <!ATTLIST import doctype CDATA #REQUIRED
                     importPath CDATA #REQUIRED
                     category CDATA #REQUIRED>
    <!ATTLIST map name CDATA #REQUIRED
                  config CDATA>
]>

<serum-jobs>

    <collection name="ticketline2">

        <imports>
            <import doctype="TICKET-NEU"
                    importPath="F:/SERUM/_inputs/use case inputs/ticketline/"
                    category="Ticketline"/>
            <import doctype="COLLAB"
                    importPath="F:/SERUM/_inputs/use case inputs/motion1.doc"
                    category="Motion"/>
        </imports>
        <maps>
            <!-- config is optional, file has to be in workingdir -->
            <map name="complete" config="ticketline-complete.prop">
                <segments>
                    <segment>Titel</segment>
                    <segment>Kurzbeschreibung</segment>
                    <segment>Auswirkungen</segment>
                    <segment>Vorbedingungen</segment>
                    <segment>Beschreibung des Ablaufes</segment>
                    <segment>Nichtfunktionale Anforderungen</segment>
                    <segment>Anmerkungen</segment>
                </segments>
            </map>
            <map name="4segments-static" config="ticketline-static.prop">
                <segments>
                    <segment>Titel</segment>
                    <segment>Kurzbeschreibung</segment>
                    <segment>Auswirkungen</segment>
                    <segment>Beschreibung des Ablaufes</segment>
                </segments>
            </map>
            <map name="complete-default"> <!--  use default properties -->
                <segments>
                    <segment>Titel</segment>
                    <segment>Kurzbeschreibung</segment>
                    <segment>Auswirkungen</segment>
                    <segment>Vorbedingungen</segment>
                    <segment>Beschreibung des Ablaufes</segment>
                    <segment>Nichtfunktionale Anforderungen</segment>
                    <segment>Anmerkungen</segment>
                </segments>
            </map>
        </maps>
    </collection>
</serum-jobs>
```