TECHNISCHE UNIVERSITÄT MÜNCHEN

FAKULTÄT FÜR INFORMATIK

Bachelor's Thesis in Informatics

Recommendation and Recovery of Traceability Links

Henning Femmer

Technische Universität München

Fakultät für Informatik

Bachelor's Thesis in Informatics

Recommendation and Recovery of Traceability Links

Empfehlung und Wiederherstellung von Traceability Links

Author:	Henning Femmer
Supervisor:	Prof. Bernd Brügge, Ph.D.
Advisor:	Jonas Helming
Submission Date:	21.10.2009

Abstract

Traceability is the ability to trace artifacts during the software life cycle. To establish traceability, artifacts are connected by traceability links. Finding these links, however, is costly and non-trivial. Different approaches addressing this problem have been presented in science, mostly trying to find similarities in the textual content. These strategies assume that all implicitly linked artifacts have similar content, which does not hold for all cases.

This thesis discusses different use cases, where a computer system can support the creation of traceability links. Requirements to this system are proposed and a framework establishing this functionality with the various strategies is presented and evaluated. Furthermore, trying to overcome shortcomings of content-based approaches, we develop and evaluate new approaches. They contain not only content-based, but also history- and structure-based algorithms. These approaches are even combined in order to combine advantages of either strategy.

Contents

Part I.

Introduction

1. Introduction

≫ *Humans have 100 different brain centres that all work in slightly different ways. You shouldn't be working on a single solution; you should be working on a host of gadgets.* ≪

Marvin Minsky on Common Sense and Computers that Emote [Rou06]

This chapter presents the motivation of this thesis; furthermore, it covers the problem description, goals and structure of this work.

1.1. Motivation

Traceability is the ability to follow an artifact through the life-cycle of a software project [GF94]. Different artifacts are connected by traceability links. This means that one can trace relations between e.g. functional requirements and use cases over the complete time. The need for traceability in software projects is widely accepted (e.g. [DP98]). Nevertheless, creating traceability is a costly and non-trivial task as most of the work to create links between artifacts needs to be done manually. Hence, many projects decide not to make the effort, resulting in the problem that developers are to rely on their own knowledge about the system [SCK+04]. Inadequate traceability plays indeed an important role concerning time problems of software projects [LFOT07]. Moreover traceability helps to identify side-effects of changes by showing links between artifacts [SCK+04].

To overcome this problem, automated or semi-automated link recommendation and recovery can be a solution. It tries to reduce the effort of establishing traceability, by suggesting certain links, so that the user is only needed in special cases or as a control instance. One can differentiate two distinct use cases:

On the one hand a user needs support (in terms of recommendations) when planning to create a certain link between two artifacts. In most cases the user would have to enter the name of the artifacts or search through a long list in order to find the correct artifacts for linking. As regular projects involve thousands of artifacts, this search can take a significant amount of time. Instead, we propose that some elements with a higher probability should be suggested on top of the list, so that in many cases the user needs only to accept the suggested elements. This use case mostly occurs right after or during creation of an artifact and is initialised by the author of this artifact. We refer to this use case as **link recommendation**.

On the other hand we need suggestions for links of a wide field of artifacts after they have been created (*after-the-fact*). This is rather initialised by the project leader and executed over entire sections of the project or over all link types of one artifact. A reason to run this can be project constraints, like the requirement to have all source code objects traced to test cases. We refer to this use case as **link recovery**. However, as link recovery is only an aggregation of many link recommendations, we will use the latter as the umbrella term.

1.2. Problem Description

Intensive research has been carried out on the area of link recovery and link recommendation (see chapter 3). Nevertheless, hardly any solution has been integrated into a complete case-tool [LOT08]. Hence nearly all scientific work focuses on links of artifacts describing the target system, like functional requirements or source code, and hardly any paper has been written about tracing links to organisational artifacts, like work items or tasks. The entirety of artifacts of the former will be referred to as the *system model*, the latter will be called the *project model* (see chapter 4). Another result is that to all our knowledge all authors trace links after-the-fact, based on the final (finished) artifacts.

Secondly all yet known approaches have their premises and thus their shortcomings [DKNH09]. For example, obviously sometimes the user wants to connect elements, which have hardly any similarity of content at all. As all information is gained by such premises, these approaches fail if the premises do not hold, i.e. a content-based approach fails, if the elements have no similar content. To overcome this problem, many authors suggest the combination of various strategies to further increase the performance of methods (e.g.: [LOT08]). To all our knowledge this has not been done and evaluated in any scientific paper in research.

A third point is, that all presented approaches rely on an artifact- or document-based representation of content, such as requirements documents or source code. No experiences have yet been shared applying common strategies in a model based approach (like the unified model), despite the fact that elements in the unified model have certain peculiarities. To name one of them, elements in the unified model might have a reduced amount of textual description, because much information does not need to be written redundantly, due to the possibility to refine elements with links.

1.3. Goals

This thesis aims to propose a system for link recommendation and recovery of artifacts of all kinds. The software system of interest should support the user in the creation of traceability links in two different ways: link recommendation and link recovery. A case-tool in the unified model (i.e. UNICASE) shall be extended to achieve this functionality.

As new research is likely to find new strategies, an extendible system is needed to guarantee the flexibility to be up-to-date and deliver strategies later-on. Furthermore the system needs to provide the option to combine different strategies in order to overcome shortcomings of the individual approaches.

1.4. Structure of this work

This work is structured into five different parts:

The first part, “Introduction”, shortly explains traceability and names the existing problems as well as the resulting goals for the thesis at hand. Furthermore deeper introductions into traceability and link recommendation are given, explaining terms and strategies and listing related work. A section describing UNICASE finishes the first part.

The second part, “Analysis”, contains the requirements for the present project. It shows in detail functional and nonfunctional requirements and depicts the targeted solution by showing certain scenarios and use cases. These are evaluated in an analysis of the resulting software system.

The third part, “Prototype”, approaches the software solution from distance to detail. The system design is shown and certain design decisions are explained. Afterwards the object design explains the source code in detail.

The fourth part, “Evaluation”, reviews the entire project and tries to measure and evaluate the success of the chosen approach. It also tries to give an outlook on which further work can and will be done for additional success of the presented work.

The last part “Appendix” lists the references from this work.

2. Introduction to Traceability Link Recommendation and Recovery

≫ *What's in a name?*
That which we call a rose
By any other name would smell as sweet. ≪

Shakespeare, Romeo And Juliet

This chapter shortly introduces into traceability presenting its benefits and impediments. It then explains traceability recommendation and recovery and how it can overcome the hindrances presented before. Also typical activities involved into the process of link recommendation and recovery are explained. To specify these rather general ideas, we then explain the various approaches used in this thesis in detail, i.e. strategies of recommendation and selection. This leads to the question, how recommendations can be compared and evaluated, which is discussed in the last part of this chapter.

The following example will clarify the different stages of a recommendation and will be used throughout this chapter for examples. Given a project about a supermarket, and that we want to connect the use case "Open Doors" with an appropriate functional requirement. To simplify the scenario, let us imagine there were only five functional requirements in this project:

1. "Door opens 'automatically' " (the correct option)
2. "Pay desk has emergency button."
3. "Doors are secured by alarm"
4. "CashRegister can be opened remotely"
5. "CashRegister is immediately sending (all) changes"

2.1. Traceability

Traceability is an important field in software engineering, many publications have been written about this topic. This section gives an overview over traceability.

2.1.1. What is Traceability?

The definition of traceability depends on which aspect one wants to emphasise:

- **Purpose-driven:** focuses on the motivation of traceability, for instance proving that requirements have been implemented.
- **Solution-driven:** focuses on how traceability is meant to be created, for instance ability of connecting artifacts.
- **Information-driven:** focuses on the artifact and the links to be created, for instance linking requirements, tests, et cetera.
- **Direction-driven:** focuses on the process of following an item through its life cycle, for instance tracing a requirement to the software component.

Gotel and Finkelstein sum up these aspects in their definition:

Definition Traceability is the ability to describe and follow the life of an artifact (requirements, code, tests, models, reports, plans, etc.) developed during the software life-cycle in both forward and backward directions (e.g. from requirements to the modules of the software architecture and the code components implementing them and vice-versa) [GF94].

Traceability is often used in the context of requirements traceability, which was introduced by the US Government's Department of Defence [GF94]. In requirements traceability functional requirements are linked to scenarios and use cases, can then be linked to classes or models and from there to tests, plans or also all kind of organisational artifacts.

2.1.2. Why Traceability?

The need for Traceability is widely accepted in academics. Besides the benefits of requirement tracing mentioned above, its profits can be used in various fields:

- **Program comprehension:** If a programmer is new to a project or system, traceability links can support the process of familiarising with the system. The programmer can look up directly which class is inserted where and for which requirement. Also the **maintenance** of legacy computer systems benefit from this assistance, especially when design analysis is performed like proposed e.g. by [CJ90].
- **Reuse of Existing Software:** Traceability from code to other artifacts helps indexing and thus searching through existing code. Furthermore can tracing code to artifacts help finding candidates for reuse [ACC$^+$02].
- **Impact Analysis:** When requirements are changed in a late phase of the project, it is important to know on which parts of the project this has an impact, for example to know which tests need to be changed. Using existing traceability links, the probable impacts can be analysed and calculated [ACCDL00].

2.1.3. What impediments to Traceability exist?

Though the need for traceability in software engineering is widely accepted [DP98], there are several impediments to the spread of traceability in real world projects.

[Wri91] identifies the lack of commitment by all parties as a major drawback, [RGP86] names the granularity of traceable entities, another reason is the missing of integration of traceability in common tools [GF94]. UNICASE overcomes most of these problems, as all artifacts are stored within one tool. The hindrance left is, that all links have to be drawn manually and personally by the user. This is a costly process, as creation of these links in bigger projects is time-consuming [SCK+04]. For this reason an automatic or semi-automatic support in the traceability process is required.

2.2. Link Recommendation and Recovery

Link recommendation and link recovery tries to solve the traceability problem by automatically or semi-automatically creating traceability links. This is a non-trivial process.

The process of link recommendation involves four major steps (see figure 2.1):

1. Selection of the artifact, from which the links starts. This artifact is referred to as the **base element**.
2. Selection of possible (and requested) artifacts: In most cases the user wants to connect only a certain type of artifacts with the base element. According to this, a list with all possible and requested elements needs to be retrieved. We refer to these artifacts as the candidate elements or **candidates**.
3. Calculation of a recommendation function: A **recommendation strategy** calculates values between zero and one for each candidate, indicating which element is how likely to be connected to the base element. We will explain these strategies in section 2.4.
4. Selection of Suggestions: Lastly a **selection strategy** needs to decide, which elements are proposed to the user and which are not. The different possibilities for selection strategies are explained in section 2.5.

Coming back to our example: We can identify that “Open Doors” is the base element and retrieved all functional requirements as candidates. Afterwards a recommendation strategy is applied on the artifacts. Imagine it had the following results:

Functional Requirement	Value Returned from Strategy
1: “Door opens ’automatically’ ”	0.55
2: “Pay desk has emergency button”	0.1
3: “Doors are secured by alarm.”	0.6
4: “CashRegister can be opened remotely”	0.3
5: “CashRegister is immediately sending (all) changes”	0.00

Table 2.1.: Link Recommendation: An Example 1

The last step is now to select some of these and suggest them to the user. For example, it does not make sense to present functional requirement #5. Assuming that we suggest all candidates above a value of 0.5, then the suggestion would be functional requirement #1 and functional requirement #3.

As the general way is now clear, it is now of interest which recommendation and selection strategies exist.

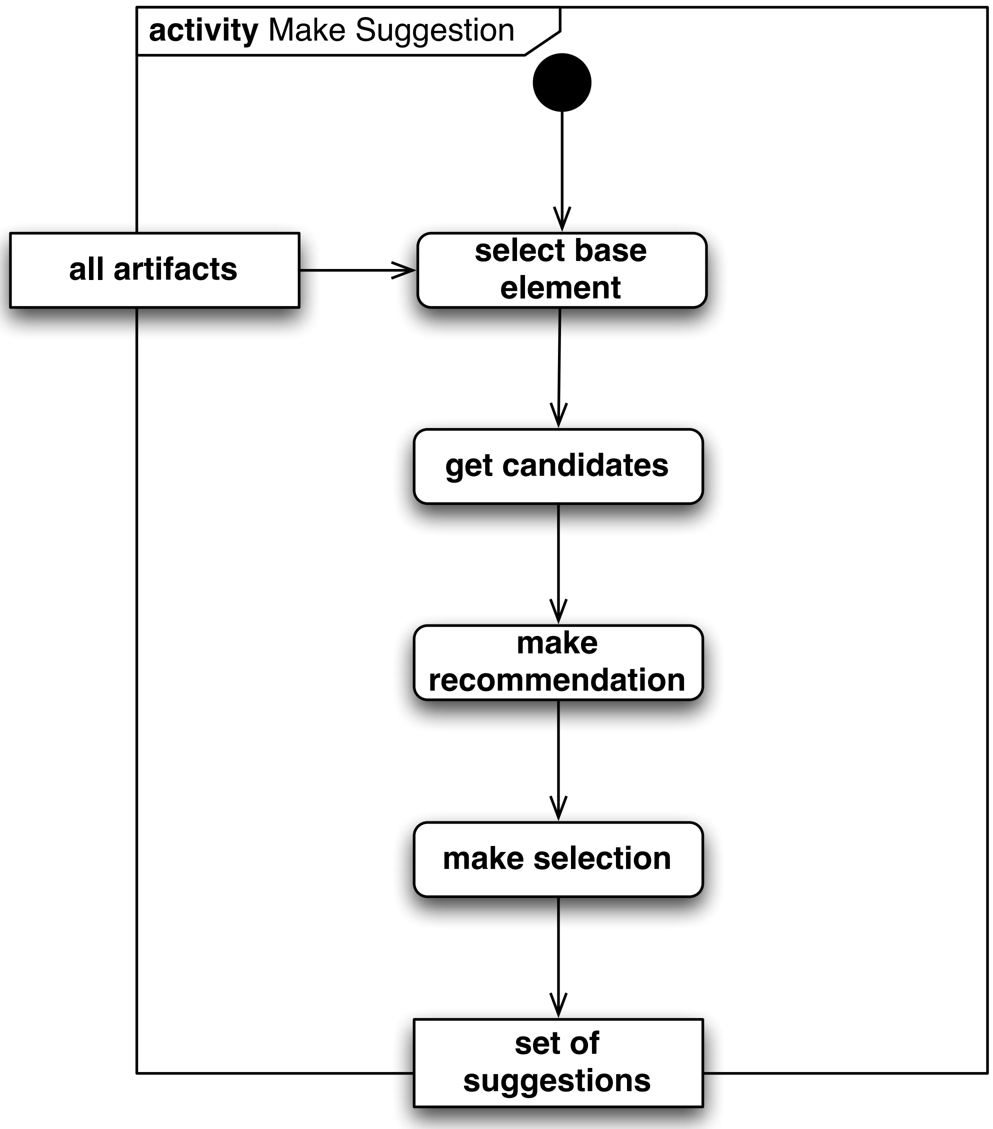


Figure 2.1.: Making a Suggestion, UML activity diagram

2.3. Information Retrieval

In the history of link recommendation and recovery very often information retrieval (IR) methods have been used to determine content similarities of artifacts. IR is concerned with representation, storage, organisation, and access to information [SCK$^+$04], for instance in typical Internet search machines or libraries. It can be traced back to 3000 BC when Sumerians created classifications for indexing and finding clay tablets [Sin01].

Assuming that related artifacts share a number of terms in their descriptions, the process of link recommendation shows remarkable similarities to information retrieval [HDS06], as both try to find certain keywords in different documents. Hence it suggests itself to use the methods of IR in link recommendation. Some strategies for text comparison in IR are: [HDO03]

- Boolean Model (BM)
- Probabilistic Model (PM)
- Vector Space Model (VSM)
- Latent Semantic Indexing (LSI)

The first, boolean model, suggests to create a binary term-to-document matrix indicating if a term appears in a certain document, and hence sorting the results by the number or percentage of hits [HDO03]. However, this model has various shortcomings due to its simplicity. For example, if a term, say “and” appears in nearly every candidate it is probably not a very important term. It furthermore does not make a difference in how often a certain term appears in both the base and candidate elements. Because of these reasons the boolean model will not be followed in the further part of this thesis.

The same refers to the probabilistic model, a different approach also working with a term-to-document matrix, but this time integer-based. It tries to calculate probabilities based on a different probability estimation techniques [Sin01]. But as various experiments showed that vector space model outperforms the probabilistic model (e.g. [ACLM99]), we will follow the remaining two strategies, VSM and LSI, which will be explained in the next section.

2.4. Recommendation Strategies

Various techniques can be used to find some sort of similarity or connection between different elements. They can be divided in content-based and non content-based strategies, dependent on whether they use the textual content for their calculations or not. Non content-based approaches use other information, like the artifact’s history or the structure of the project, and the position of this artifact within the project.

2.4.1. Content-based Strategies

Content based approaches extract the textual contents of elements and try to compare these texts. However, to count and compare words from a text, these so called keywords need to be extracted first.

Preprocessing: Keyword Extraction

All content based approaches need to select, extract and normalise the textual content from the underlying data. In order to retrieve as much information from the text as possible, we need to transform words in different grammatical forms to one single term. For instance *recommendation*, *recommended* and *recommend* need to be transformed to one single term. This transformation is called stemming.

The extraction process consists of the following stages (see figure 2.2):

1. Extract text from artifact
2. Replace *stop chars* (like '/', ',' etc.) with empty spaces
3. Split the text by spaces
4. Go over each word and:
   a) Check for *stop words*, e.g. "and", "is", as those should not alter the results
   b) Split camel-case words
   c) Make it all lower case letters
   d) Stem the words, i.e. try to reduce all forms of a word to one singular form

This keyword extraction must be executed in all content based approaches prior to calculation. For stemming we applied a very basic (and thus fast) algorithm. Noteworthy is also that only words with a certain length have been stemmed to prevent destroying abbreviations. The stemming algorithm replaces certain endings, according to the following table (replaced by ϵ means replaced by nothing):

Suffix	Replaced by	Example
sses	ss	possesses → possess
ied	y	identified → identify
ies	y	identifies → identify
ed	ϵ	walked → walk
s	ϵ	walks → walk
ing	ϵ	walking → walk, meeting → meet
ings	ϵ	meetings → meet
ion	ϵ	encryption → encrypt
ment	ϵ	astonishment → astonish

Table 2.2.: Stemming: List of Suffixes and Replacements

Going back to the example: Functional requirement #5 is going to show how the process works.

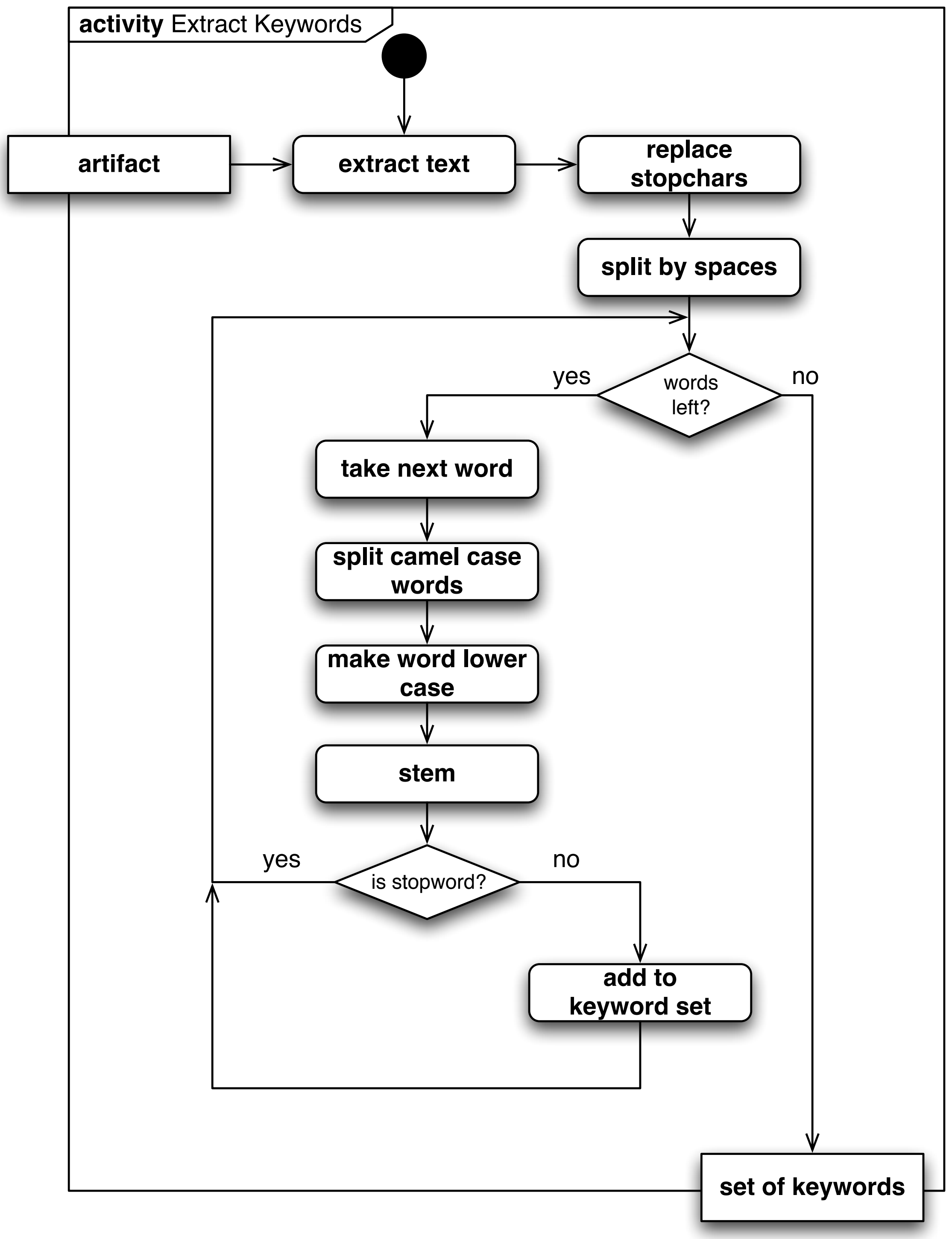


Figure 2.2.: Extracting Keywords, UML activity diagram

Step	Words (separated by commas)	Note
0	CashRegister is immediately sending (all) changes	initial text
1	CashRegister is immediately sending all changes	removed brackets (stop chars)
2	CashRegister, is, immediately, sending, all, changes	split
3	cash register, is, immediately, sending, all, changes	processing *CashRegister*
4	cash register, immediately, sending, all, changes	processing *is* (stop word)
5	cash register, immediately, sending, all, changes	processing *immediately*
6	cash register, immediately, send, all, changes	processing *sending*
7	cash register, immediately, send, all, change	processing *changes*

Table 2.3.: Link Recommendation: An Example 2

Vector Space Model (VSM) with tf-idf-weighting

Studies (e.g. [ACLM99]) have shown that from the basic approaches the Vector Space Model (VSM) is the most useful. For this strategy a list of words is referred to as a “document”, so each candidate is transformed into a list of terms. We assume that these terms are already preprocessed in the way mentioned above. To calculate suggestions in VSM a term-by-document matrix is created, counting the occurrences of each term in each document. Now two transformations are made on this document to diminish the following problems:

1. Some documents contain many words, some only a few; nevertheless should both be equally treated.
2. Some words happen to appear in many documents. These are probably overseen stop words or words that occur frequently in certain application domains. These words should not have a high impact on the overall result.

These problems are dealt with by **tf-idf-weighting**, which is computed as following: For each calculation there is a dictionary containing all words of all documents. Each document d consists of a set of values $< v_1, v_2, \ldots, v_N >$ containing one value for each term from the dictionary. The values are then calculated by

$$v_i = tf(i,d) * idf(i)$$

- with $tf(i,d)$ denoting the term frequency: The number of occurrences of the i-th term of the dictionary in document d, divided by the number of terms appearing in d and

- with $idf(i)$ denoting the inverse term frequency mostly calculated by

$$idf(i) = \log_2(\frac{D}{df(i)})$$

- where $df(i)$ counts the number of documents in which term i appears and D the number of documents in total [HDO03].

However some authors suggest to use different weighting functions, like Okapi-Weighting or Pivoted Normalisation Weighting [Sin01]. No matter how the terms are weighted, finally all documents are collected in a number of vectors or a matrix containing the weights. The similarity between two documents can now be easily calculated as the cosine between the referring vectors in the matrix. As all values are positive within the created matrix, the cosine is always a value between 0 and 1 describing the difference of direction of the vectors. Moreover the cosine can be calculated by the normalised dot product of these vectors:

$$\cos(a, b) = \frac{\sum_{i=1}^{N} a_i * b_i}{\sqrt{\sum_{i=1}^{N} a_i^2 * \sum_{i=1}^{N} b_i^2}}$$

Latent Semantic Indexing (LSI)

Latent Semantic Analysis is a IR technique first proposed by Deerwester et alt in [DDF$^+$90]. In IR it is widely known under the term Latent Semantic Indexing (LSI). In LSI the term-document-matrix is taken and decomposed with Singular-Value-Decomposition (SVD). This results in the three matrixes

$$A = U * S * V^T$$

where U's columns are the left eigenvectors, S' diagonal entries are the singular values and V's columns are the right eigenvectors of the Matrix A. The singular values in S can be seen as the impact certain parts of the matrix have. In fact, these parts can be seen as some “latent semantic” structure underneath. Keeping just the k strongest values in S and equally in U and V, one can construct what is called an LSI subspace:

$$A_k = U_k * S_k * V_k^T$$

This resulting subspace can be seen as a noise reduction of the former matrix, as the weakest factors are eliminated. Latter the new matrix is queried via the cosine, similar to documents in VSM. Nevertheless the selection of the factor k is still an open issue in science [LD06], and also the calculation of the SVD is a computationally complex task.

2.4.2. Non content-based Approaches

Content-based strategies depend very much on the content description of the data the system is working on. However, sometimes connections between elements can not be found in the textual content, so other possibilities have to be developed. One of them is trying to find similarities between objects in the change behaviour in the past (**history-based approaches**), others try to find connections over the project's structure (**structure-based approaches**).

2.4.2.1. History-based Strategy

History-based approaches access the project's history for information.

Association Rule Mining (ARM) Association Rule Mining is a technique to find artifacts frequently changed together. It is based on the idea, that artifacts changed together have some sort of common content. The versioning system of UNICASE can be used as a database for ARM: Whenever a new set of changes (*change package*) is applied to the project, they are iterated and a value is calculated, determining how far away the elements are in the change list. This value is saved or added to the already existing value for this combination of artifacts. Accordingly, when a suggestion has to be made, the strategy looks up this value for each combination of base element and candidate. After retrieving all values from the database, they are normalised, so that all values are $\in [0, 1]$. Based on experiences in [DKNH09], we limited the distance of two changes of elements to a maximum of ten, all changes of a bigger distance were ignored.

2.4.2.2. Structure-based Strategies

In contrast to content-based approaches and the history-based approach we developed structure-based approaches. They look at the project in its entirety (SRR) or compare connections of certain special links (RAR).

Related Assignees Recommendation (RAR) A developer has many action items assigned to him. The Related Assignees Recommendation (RAR) assumes that a functional requirement is mainly assigned to a single user. Hence, whenever an action item of a certain user needs to be connected to a functional requirement, those items the user already worked on, are more probable candidates than others (see figure 2.3).

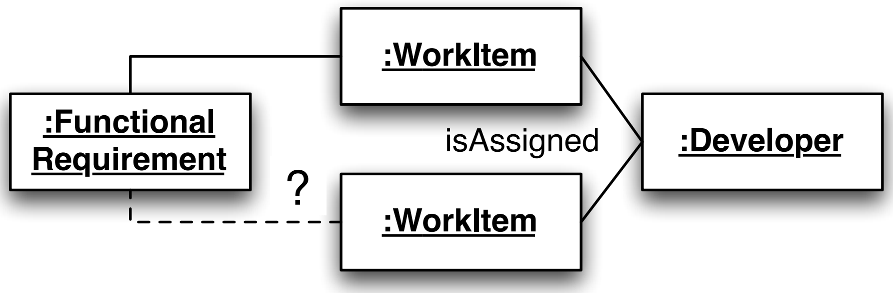


Figure 2.3.: Action Items, Developers and Functional Requirements, UML class diagram

This approach only works for action items, which are already assigned to a developer. Starting from the assigned developer, we consider all other action items, which are assigned to the developer. We count all links from these action items to functional requirements and normalise this number. This is accomplished by dividing all counts by their local maximum. This leads to a suggestion between 0 and 1 for all linked functional requirements.

Shared References Recommendation (SRR) The shared references recommendation is based on the assumption that related artifacts are linked to an intersecting set of other artifacts. Imagining the unified model as a graph of all connected elements in both system and project model, we can identify clouds around artifacts, containing all linked artifacts recursing to a certain depth (see figure 2.4).

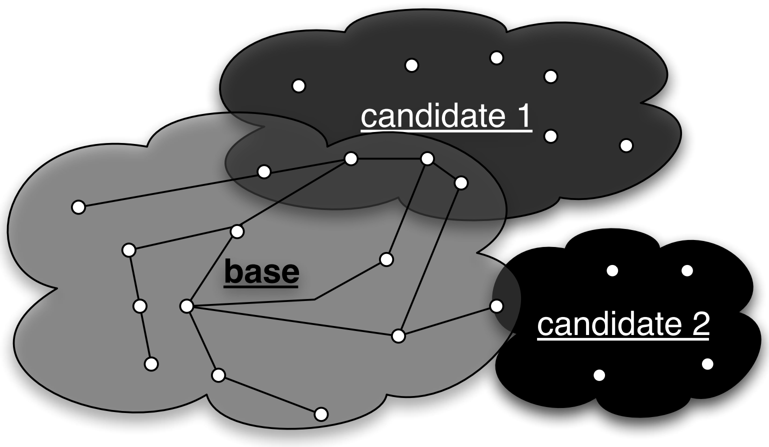


Figure 2.4.: Semantic Clouds in the SRR approach

During development we reckoned that elements whose clouds have a greater intersection are more likely to be connected to each other; consequently, we create clouds for each candidate and see if they intersect, and if so, how strongly. Again, the result is normalised by the maximum. The question of how to limit the size of the clouds will be evaluated in the appropriate section. We suggest two possibilities: either limit the cloud by radius (through the depth of iteration) or by size i.e. through the number of elements.

2.4.3. Combinations

There are a couple of possible solutions for the combination of different recommendation strategies. We suggest combining the normalised double values from different strategies with a mathematical function. The further question is which mathematical function to use. The first possibility is to take the maximum. The idea behind this proposal is the assumption that the return of a high value by a certain strategy indicates evidence of a link between two elements. The other option is to weight each strategy with a certain ratio. This leads to the most natural combination: Calculating the arithmetic mean. However other ratios might be adequate, for example when it is known that a certain strategy produces very high values.

2.5. Selection Strategies

As discussed before, the recommendation process contains four steps. The last two of those involve a recommendation strategy and selection strategy. After the various recommendation strategies were explained in the last section, this section describes the different selection strategies.

After all probabilities or values for a certain link have been calculated, one needs to decide which links are selected for the recommendation. This action is known as link selection. [LD06] categorises the possible strategies based on [LFOT04]:

- Absolute threshold: All suggestions above a certain probability T are selected.
- Relative threshold: All suggestions above a certain probability ϵ are selected, but in this case ϵ is relative to the metric used, e.g. the highest 30% of the values of a metric.
- Scale threshold: All suggestions above a certain probability ϵ are selected, with ϵ this time scaled between 0 and the maximum similarity M, i.e.

$$\epsilon = c * M (0 \leq c \leq 1)$$

- Cut point: The topmost x elements are selected.
- Cut percentage: The topmost x% of all elements are selected.

Coming back to the example: Assuming the recommendation strategy returns the results from table 2.1, taking the selection strategies from above and some random example parameters, they would suggest the following functional requirements for creating traceability links:

Selection Strategy	Parameters	Suggestion
absolute threshold	threshold=0.5	#1, #3
relative threshold	highest 40%: values over 0.6	#3
scale threshold	$c = 0.4$: values over $0.4 * 0.6 = 0.24$	#1, #3, #4
cut point	cp=1	#3
cut percentage	percentage=50%	#3, #1, (#4-boundary case)

Table 2.4.: Link Recommendation: An Example 3

2.6. Evaluation of Link Recommendation

After using a recommendation and a selection strategy, the question is now, how to analyse these recommendation and suggestions strategies properly, in order to find out which strategy is the best. Furthermore do we need exact measurement to qualify the different strategies.

2.6.1. Methods of Evaluation

Given some strategies for recommendation and selection, we want to know how good our approaches are. To achieve this, we need comparison data to say which link would be suggested in a certain suggestion and which of those would have been a right suggestion. Retrieving this data can be done in two possible ways:

1. Questioning of users
2. Simulation with data from recent projects

In option one, a group of people are recorded while using the software. Whenever the link recommendation feature is used, one needs to store which element the user wanted to connect and which elements were suggested. The problem of this approach is that it is costly and time consuming: For every single strategy we need a significant number of test cases and users, who execute the strategies. Another problem is that all the users in the tests need to know the projects and data used to make sure that the comparison data they produce is valid.

Because of these shortcomings we used the second method: Existing projects are taken and analysed with the links created in these projects. This can be either done by taking the last state of the project (**state-based evaluation**) or by recreating the project using the project history and making decisions at the time of creation (**history-based evaluation**). Regarding the former, for all existing links under consideration, suggestions are made and compared with the real existing links. Regarding the latter, for every state of the project, where a link was created, at that very point a suggestion is made and compared with the created link. Further details on which links we measured in what way, will be given in chapter 10.

Now that we have comparison data with correct links and suggested links, we have to find measurements to compare these values for different strategies. One example: If a strategy always suggests all artifacts for creating links, it will always find every possible link. But this has the problem of creating to many incorrect links. On the other side, if never suggests any artifact, it will never make the mistake of suggesting incorrect links, but neither will it find any correct link. To measure these contrary aspects, two metrics are widely used in science.

2.6.2. Precision and Recall

Precision and recall are common measurements for efficiency in Information Retrieval contexts. They are widely used for comparing the quality of certain approaches [MM03].

Precision

Precision is a rate of exactness. It is a number $\in [0,1]$ calculated by the number of correctly proposed elements divided by the total number of suggested elements, i.e.:

$$Precision = \frac{|\text{RelevantElements} \bigcap \text{SuggestedElements}|}{|\text{SuggestedElements}|} \tag{2.1}$$

In the context of the present thesis, to get the precision we calculate a value for each test:

$$Precision = \frac{|\text{CorrectLinks} \bigcap \text{SuggestedLinks}|}{|\text{SuggestedLinks}|} \tag{2.2}$$

The resulting value is a probability of how certain a user can be selecting a random suggested element that this element is what he[1] wants.

[1]For pure readability reasons we will furthermore stick to the generic masculine.

Recall

On the contrary, recall is a measurement for completeness. It is used to estimate the question, whether an algorithm selects all relevant elements. Thus the ratio is calculated via:

$$Recall = \frac{|\text{RelevantElements} \bigcap \text{SuggestedElements}|}{|\text{RelevantElements}|} \tag{2.3}$$

or in our case:

$$Recall = \frac{|\text{CorrectLinks} \bigcap \text{SuggestedLinks}|}{|\text{CorrectLinks}|} \tag{2.4}$$

The result of this measurement is an indicator for how certain a user can be hat, if he selects all the system's suggestions, all information possible is retrieved.

Example

Coming back to the example: We will examine calculation of precision and recall on the example suggestion made in table 2.4 using equations 2.2 and 2.4. Remember, that functional requirement #1 was the correct element.

Suggestion	Precision	Recall
#1, #3	0.5	1
#3	0	0
#1, #3, #4	$\frac{1}{3}$	1
#3	0	0
#3, #1, #4	$\frac{1}{3}$	1

Table 2.5.: Link Recommendation: An Example 4

3. Related Work

≫ *It has all been the work of others.* ≪

Otto Hahn, father of nuclear chemistry and Nobel Price winner

A lot of work has been written recently about link recovery. This chapter lists results of recent and contemporary papers. They differ not only in the certain strategies, but also in the fields in which the strategies were applied.

All strategies try to measure the relevance of the candidates to the base element. To recommend or find links automatically between these objects, a certain connection has to exist. Different approaches make different assumptions about these similarities. Content-based approaches argue that linked documents have equalities in their textual content. History-based approaches belief that interrelated artifacts are mostly changed together. Other strategies analyse the structure of a project.

The following list of sources is doubtless not exhaustive. For a great overview over the topic see [LFOT07].

3.1. Content-based Approaches

Most of the content-based approaches try to measure some sort of similarity between different documents. In recent time very often Information Retrieval (IR) techniques have been used to determine this value. Therefore certain describing keywords are either manually entered by the user or extracted from related content and context.

From IR context mainly three different methods are of interest: Probabilistic Model, Vector-Space-Model and recently Latent Semantic Indexing.

3.1.1. Basic Approaches

In an early paper, [SSC96] try to determine mismatch between design and code by applying reverse engineering and logic-based static analysis. A certain visualisation is created providing help to determine these mismatches.

Murphy et al. researched in [MNS01] on so called software reflexion models trying to trace source code to design. A five-step iterative procedure is used, creating regular expressions to find links between design artifacts and c++ source code. It also mentions the potential of structural information.

[Bri03] is especially focused on change analysis. Their work includes Unified Modelling Language (UML) consistency checks and Open Constraint Language (OCL) based change determination between different document versions. They furthermore present a prototype providing this functionality.

3.1.2. Probabilistic and Vector Space Model (VSM)

Antoniol et al. compare in [ACC$^+$02] the probabilistic and vector space model for tracing links from code to textual representations, namely c++ code to manuals and java code to functional requirements. According to their results these approaches lead to approximately similar results. Their studies also suggest that the probabilistic model leads to better results in cases, in which almost any of the candidate's keywords is linked to the target document; hence, VSM is more successful in examples with smaller numbers of relevant keywords. According to work in [ACLM99], [ACCL00] and [ACC$^+$00a] VSM performs better than PM in the majority of cases.

The same results follow from [ACCDL00], but suggesting that probabilistic model is a little more unsteady with respect to the input, whereas VSM seems to be rather immune against labile input. Within this work the authors analyse the change impact referring to a library of c++ code for Efficient Data types and Algorithms (LEDA).

Settimi et al. try to broaden the application of VSM on link recovery between requirements, code, tests and also UML models in [SCK$^+$04]. They also compare standard cosine and pivot normalisation weighting schemes and make use of thesauri, all with limited success.

3.1.3. Latent Semantic Indexing (LSI)

Marcus and Maletic discuss in [MM03] the use of Latent Semantic Indexing (LSI) for link recovery from documentation to source code. According to their work LSI performs as good as the methods proposed by Antoniol et al., e.g. in [ACC$^+$02], but does not need as much preprocessing, or 'stemming'.

Lormans and van Deursen apply LSI in [LD06] on links between requirements and design artifacts and between requirements and test-cases. They discuss different link selection strategies (see chapter 2) and proposed a list of open research issues.

In [HDS06] Huffman Hayes et al. further analyse their tool RETRO using LSI and VSM comparing precision, recall and a measurement building a median between those two, called f-measure. They also use user feedback, which furthermore improves the results. Links between different requirements are traced. Contrary to [MM03] in their results VSM performs better than LSI.

One of the rare examples of a real implementation of link recovery in a real case tool was proposed by De Lucia and others in [LOT08] for a tool named ADAMS, using Latent Semantic Indexing. They conclude that to fully establish a proper and complete link recovery a combination of methods is inevitable.

David [Dav08] presents an approach based on recurrent neural networks (RNN) and LSI to estimate changes. It also uses the project history to recommend related items.

3.1.4. Extensions

There are two main content-based extensions to the above mentioned models. Some authors try to improve their methods with thesauri, like [SCK$^+$04]. Huffman Hayes et al. also improve VSM with a thesaurus in [HDO03] and use these algorithms for a tracing process from high-level to lower-level requirements. Their results show that thesauri can significantly improve the present algorithms in terms of precision and recall. However, these improvements heavily impact on the performance. In [HDSH04] they integrate these results into a special purpose tool RETRO (REquirements TRacing On-target).

Other approaches use user feedback for further improvements. It is shown in e.g. [HDS06] or [ACC00b], that this can improve the precision value.

3.1.5. Problems of Content-based Approaches

Content-based approaches suggest that all links are similar in their content. While this might hold for some link candidates, others might not have any similar content at all [DKNH09]. Hardly anyone of the above mentioned authors doubts this fact, although a combination of approaches was not yet presented.

3.2. Structure-based Approaches

Non content-based approaches rely on the existing structure of the project.

Von Knethen et al. build a tool for impact analysis for different kinds of artifacts in [KGFK03]. They use assumptions over project structures and name matching to find links. Cleland-Huang et al. introduce in [CSDZ05] three features into the probabilistic model, which try to make use of hierarchy and clustering. Their work shows that different data sets can lead to very diverse results, depending on the internal structure of the data.

3.3. History-based Approaches

History-based approaches use the change of elements over time to receive information about the relations between documents. Briand et al. are in [Bri03] focused on change analysis. Their work includes Unified Modeling Language (UML) consistency checks and Open Constraint Language (OCL) based change determination between different document versions. They furthermore present a prototype providing this functionality. Various approaches, e.g. [GHJ98], [GJK03] or [YMN^{+}04] use Content Versioning Systems (CVS) for source code to source code tracing. Some approaches, e.g. [DKNH09] use association rule mining. Zimmermann et al. use in [ZWDZ05] a number of heuristics to trace textual content to object models. [oDRC^{+}02] traces requirements to text, suggesting to automatically and continuously analyse the flow of incoming events.

4. UNICASE

≫ *Union gives strength.* ≪

Aesop's Fables, The Bundle of Sticks

Software projects are usually supported by Computer Aided Software Engineering (CASE) tools. CASE tools help design, model and document software projects; they can furthermore keep information shared and synchronised in the team.

UNICASE [BCHK08] is a CASE tool to support traceability and artifact management during the entire software life cycle. This chapter introduces into the relevant parts of UNICASE for this thesis. Interested reader should refer to the UNICASE developer page for further information[1].

4.1. What is UNICASE?

UNICASE is a CASE-Tool integrating models from the different development activities (such as requirements, use cases, UML models, schedules, bug and feature models) into one unified model. This unified model is highly traceable by design. The UNICASE client allows to view and edit these models in a textual, tabular and diagram visualisation. The models are stored and versioned on a server comparable to Subversion (SVN) but customised for models. Client and server are easily extensible to support integrating new models into the unified model. UNICASE is based on the Eclipse platform including EMF and GMF. It can also be used as a framework to build modelling applications that reuse its repository and visualisation capabilities. The project is open-source and released under the Eclipse Public License v 1.0 (EPL) [Aut09b].

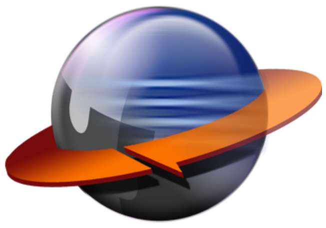


Figure 4.1.: UNICASE Symbol

UNICASE is able to store any kind of software artifacts. It is model-based, thus giving the possibility for textual description and establishment of links between nearly any kind of artifacts (referred to as **model elements**). New types of model elements can be added if needed. Furthermore, a user can walk through the entire tree of model elements following the links given. Noteworthy for our case is that UNICASE allows to access big sets of data in terms of content, structural and history information, thus giving the possibility to use this data in various methods and approaches for link recommendation and recovery.

[1]http://www.unicase.org

4.2. The UNICASE model

UNICASE contains three parts: [Aut09a]

1. **Unified Model**: The data model and data base for UNICASE
2. **Unified Repository**: The versioning and synchronisation system of UNICASE
3. **Unified Views**: The different graphical interfaces for a UNICASE project

4.2.1. Unified Model

The UNICASE model consists of two different parts, the **system model** and the **project model**.

The system model can be understood as the entirety of all elements describing the system under construction, for example classes, diagrams, functional requirements et cetera. In contrast, the project model is the sum of all organisational elements related to a project, e.g. work items, meetings, bug reports and similar artifacts.

Every element in the unified model extends the class `ModelElement`. For readability reasons, we left this out in the following diagrams. The elements in the unified model include all kind of artifacts, like use cases, functional and nonfunctional requirements, bug reports etc. Of interest for this thesis are especially the model element types `Functional Requirement`, `Use Case`, `Action Item` and `OrgUnit`, because they will be used in the evaluation in chapter 10. The following sections and diagrams shows their part in the unified model and the connection between them.

Use Cases to Functional Requirements

On the one side there is a link between functional requirements and use cases: A functional requirement can be detailed in a number of use cases. A use case describes a part or an entire functional requirement (see figure 4.2). These elements are part of the system model, as they only describe the system under construction and not organisational elements.

Work Items to Functional Requirements

On the other side work items, such as action items, issues, and bug reports, can be annotated by every type of model elements. As functional requirements are also model elements, every functional requirement can annotate a number of work items (see figure 4.3).

Work items can be assigned to so called `OrgUnits`, which is usually a single stake holder or developer in the project.

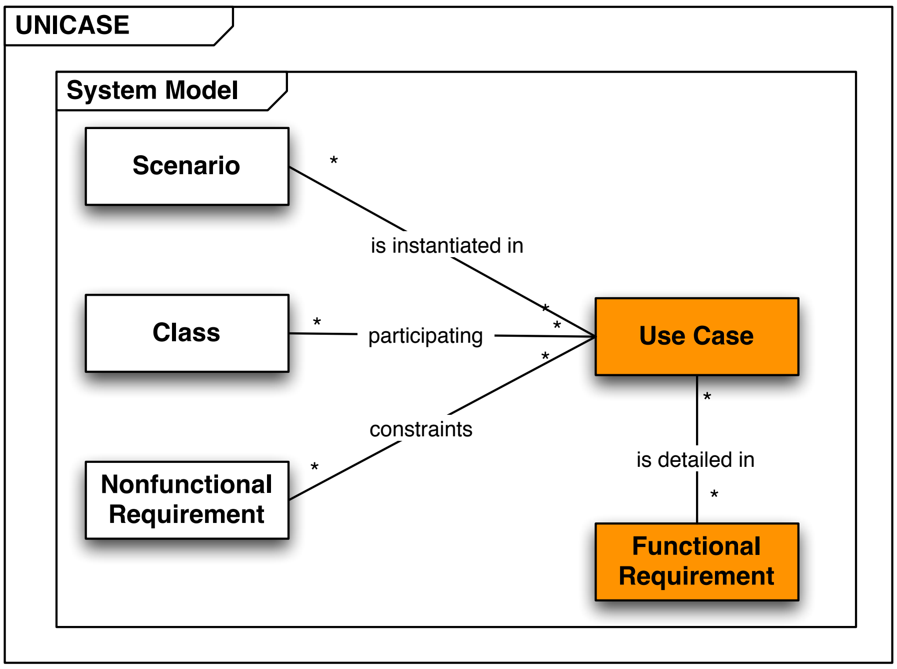


Figure 4.2.: UNICASE: Use Cases to Functional Requirements, UML class diagram

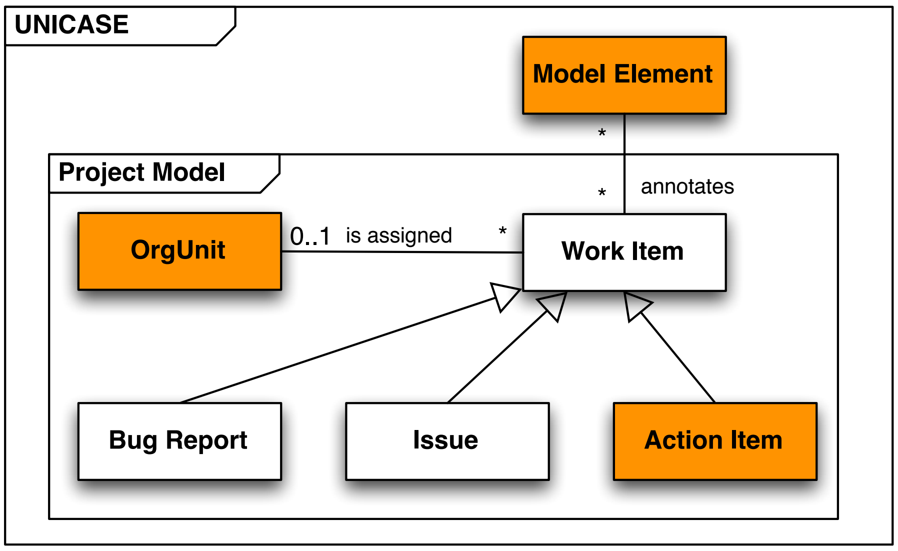


Figure 4.3.: UNICASE: Work Items to Functional Requirements, UML class diagram

4.2.2. Unified Repository

UNICASE includes the unified repository, which is a server node for content versioning of the unified model. It allows to store and update any instance of the unified model, independent from its detailed specification. This means that change in the model needs not changing the implementation of the repository. The repository features possibilities like updating, committing and merging like any other versioning system.

Changes are delivered as operations in change packages. For this thesis only reference operations, which indicate creation of traceability links, are concerned. All relevant information can then be retrieved from these operations (see figure 4.4).

The server node in UNICASE is called the **EMFStore**.

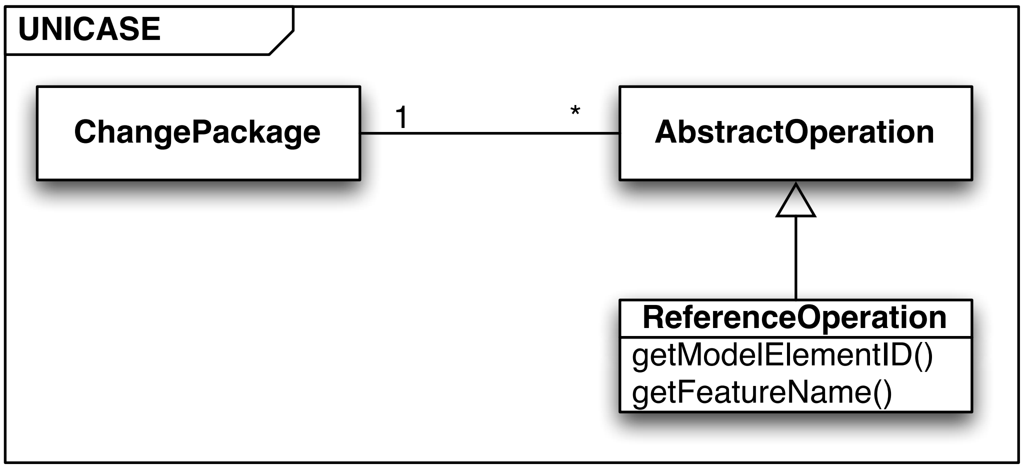
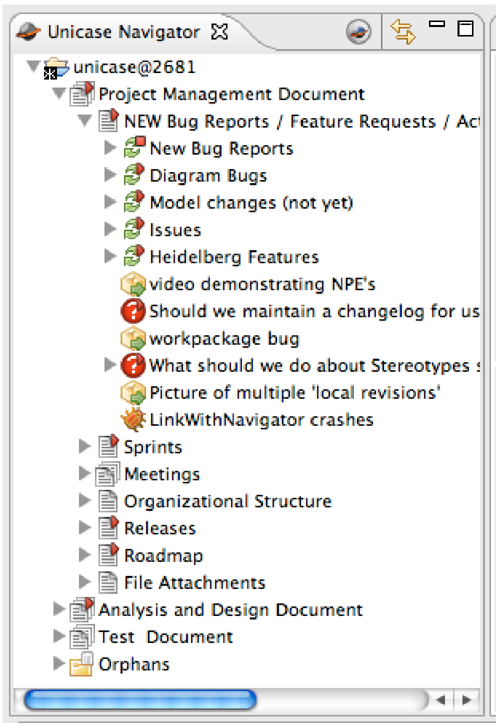


Figure 4.4.: UNICASE: Changes in UNICASE, UML class diagram

4.2.3. Unified Views

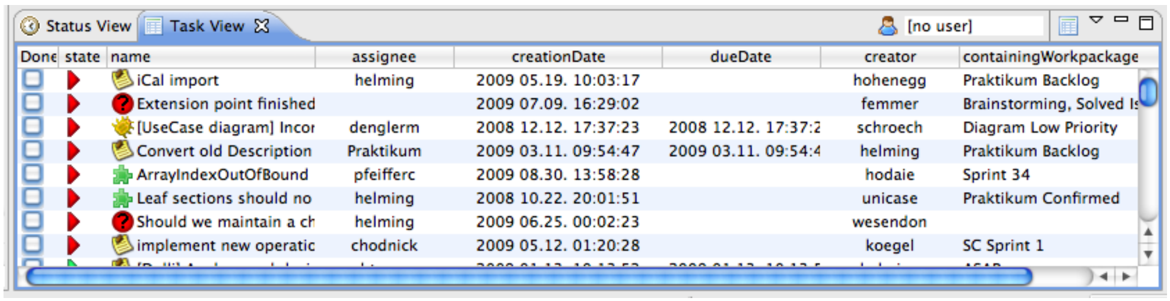
UNICASE views work independent from the specific model in use. UNICASE is delivered with various browser for basic functionality, for example the navigator (see figure 4.5) for browsing through the project and the model editor for modification of single model elements. It furthermore offers more sophisticated functions like graphical draw panels. Additional views can be easily developed and added as plugins.

UNICASE also offers special views for different roles in the project, for example a tasks view (see figure 4.6) for developers. This view offers functionalities to overview work items and sort them by assignees, dates or status. The developer can also mark tasks as finished. Another view is the status view (see figure 4.7) for project managers. Here a person can get an overview over the process of a work package, like a sprint in agile methods. The view shows how many work items have been solved, and sums up estimates entered for these work items by the developers.



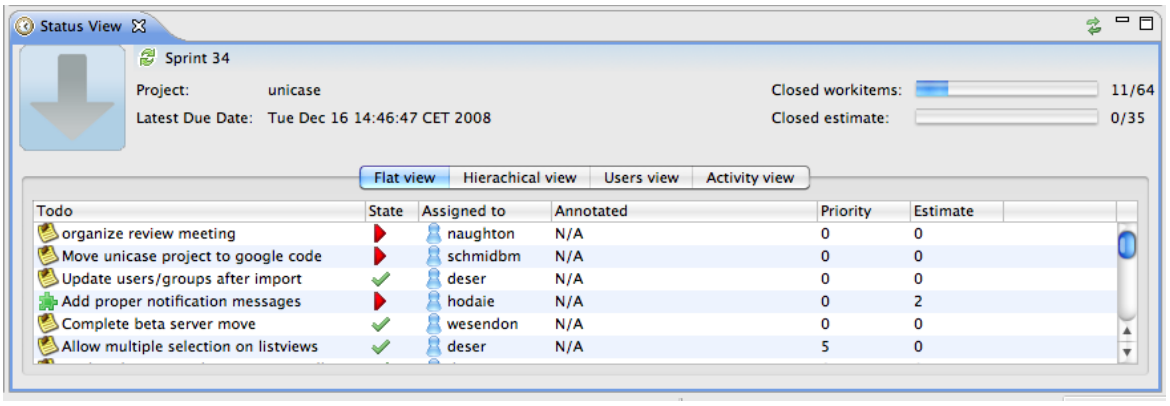
- unicase@2681
  - Project Management Document
    - NEW Bug Reports / Feature Requests / Act
      - New Bug Reports
      - Diagram Bugs
      - Model changes (not yet)
      - Issues
      - Heidelberg Features
      - video demonstrating NPE's
      - Should we maintain a changelog for us
      - workpackage bug
      - What should we do about Stereotypes
      - Picture of multiple 'local revisions'
      - LinkWithNavigator crashes
    - Sprints
    - Meetings
    - Organizational Structure
    - Releases
    - Roadmap
    - File Attachments
  - Analysis and Design Document
  - Test Document
  - Orphans

Figure 4.5.: UNICASE: Navigator



Done	state	name	assignee	creationDate	dueDate	creator	containingWorkpackage
☐	▶	iCal import	helming	2009 05.19. 10:03:17		hohenegg	Praktikum Backlog
☐	▶	Extension point finished		2009 07.09. 16:29:02		femmer	Brainstorming, Solved Is
☐	▶	[UseCase diagram] Incor	denglerm	2008 12.12. 17:37:23	2008 12.12. 17:37:2	schroech	Diagram Low Priority
☐	▶	Convert old Description	Praktikum	2009 03.11. 09:54:47	2009 03.11. 09:54:4	helming	Praktikum Backlog
☐	▶	ArrayIndexOutOfBound	pfeifferc	2009 08.30. 13:58:28		hodaie	Sprint 34
☐	▶	Leaf sections should no	helming	2008 10.22. 20:01:51		unicase	Praktikum Confirmed
☐	▶	Should we maintain a ch	helming	2009 06.25. 00:02:23		wesendon	
☐	▶	implement new operatic	chodnick	2009 05.12. 01:20:28		koegel	SC Sprint 1

Figure 4.6.: UNICASE: Task View



Sprint 34

Project: unicase

Latest Due Date: Tue Dec 16 14:46:47 CET 2008

Closed workitems: 11/64

Closed estimate: 0/35

Todo	State	Assigned to	Annotated	Priority	Estimate
organize review meeting	▶	naughton	N/A	0	0
Move unicase project to google code	▶	schmidbm	N/A	0	0
Update users/groups after import	✓	deser	N/A	0	0
Add proper notification messages	▶	hodaie	N/A	0	2
Complete beta server move	✓	wesendon	N/A	0	0
Allow multiple selection on listviews	✓	deser	N/A	5	0

Figure 4.7.: UNICASE: Status View

Part II.

Analysis

5. Requirements

≫ ***Revan:*** *Whoa, slow down there. Yes, I did purchase you...*
HK-47: *Explanation: Then you qualify as my master and I must refer to you as such. The legal requirements for models of my type are very specific, master.*
Revan: *What legal requirements do you mean?*
HK-47: *Answer: Simply that the distinction between 'killer' and 'killee' be a clear one. I cannot kill of my own volition, naturally.*
Revan: *I don't think 'killee' is a word.*
HK-47: *Expletive: Damn it, master, I am an assassination droid... not a dictionary!* ≪

Star Wars: Knights of the Old Republic

This chapter describes the requirements to this system following the FURPS [Gra92] model and quality attributes suggested by [BD03]. The first section describes the functional requirements of the system. The second section describes the quality attributes and constraints (nonfunctional requirements). All requirements are meant to be verifiable.

5.1. Functional Requirements

The link recommendation system is able to support users in the creation of new traceability links between model elements. This support is split in mainly two different use cases:

Whenever a user is working on a model element and wants to create a new link for a model element, all possible candidate elements are listed. However, more probable elements are presented more prominently, for example ranked according to a certain probability. If the presented items do not suit the user, he can also search for items by name.

The link recommendation system can also help the user in creating traceability by recommending to establish certain links. Whenever the user activates this functionality, the system checks for traceability links, which it would suggest, and notifies the user about the suggestion. Afterwards the user can decide to accept or decline the presented suggestions.

The system furthermore offers the option to search through a single model element in order to find uncreated, but very probable links. The user can start this functionality when working on a model element. Afterwards the system presents possible links and asks the user to accept or decline them.

5.2. Nonfunctional Requirements

5.2.1. Interface

The system needs to provide functionality for the UNICASE project. Hence it needs to entirely work on the data model provided by UNICASE.

5.2.2. Implementation

An implementation of this system has to be developed in java using Eclipse as integrated development environment (IDE). Furthermore must the system be an extension to the existing status of the UNICASE project; hence, code quality rules given by the UNICASE team apply[1].

5.2.3. Usability

The main constraint in usability is that the suggestion of links may not impact on the usability. This means that no explanation is necessary for a user to make benefit of the system; moreover, a user not knowing of this system must not be impaired. Additionally the user must be able to accept the system's suggestions at any time with just one click.

5.2.4. Reliability

The reliability of UNICASE must not be biased by the system. This holds also for third-party-extensions to this system as far as possible: If a developer writes and delivers a non-working plugin to the link recommendation system, it should not crash the UNICASE system.

5.2.5. Performance

The system can make a single suggestion over 3600 elements under 3 seconds. This is based on the number of candidates of projects (e.g. DOLLI II or UNICASE, see chapter 10) using the UNICASE system. The average time needed to add a link should be lower using the system than without.

Moreover, the suggestions a system makes for link recovery have to be very accurate. This is very important due to the fact that the user will not use the system, if it does not provide a certain accuracy. Thus for link recovery the strategy with the highest precision should be chosen.

In link recommendation the precision is not as important, as the system will always suggest a fix number of elements on top of the rest and hence the precision stays approximately the same. The only question is here, if the correct model element is visible on the first sight or not (which is: is the model element under this fix number of elements?). If it is not, the user will rather search for the element by name than scrolling though a list of potentially hundreds or thousands of items. Thus, the recall is very important and the technique with the highest recall needs to be selected. These considerations require evaluation of the available strategies.

[1]Which can be found under `http://www.unicase.org`

5.2.6. Supportability

The system should offer the functionality to change the strategies used for recommendation. Furthermore, combination of strategies should be possible. Lastly, any developer firm with the UNICASE project should be able to write a strategy and hereby improve the quality of recommendations for the specific project.

5.2.7. Operation

No administration should be needed after the plugin is delivered.

5.2.8. Packaging

The system is delivered the same way as UNICASE, which is through the Eclipse installation dialogue. The UNICASE system can be delivered with or without link recommendation.

5.2.9. Legal

The resulting code will be delivered open-source and should therefore be licensed under Eclipse Public License (EPL) 1.0[2]. This includes the constraint that no copyright protected frameworks is allowed to be used for implementation.

[2]To be found under `http://www.eclipse.org/legal/epl-v10.html`

6. Analysis System Models

≫ *An absolute can only be given in an intuition, while all the rest has to do with analysis.* ≪

Henri Bergson, French philosopher and Nobel Price winner

This chapter indicates scenarios of usage of the link recommendation and recovery system. On this basis use cases are identified and an abstract object and dynamic model is suggested.

6.1. Scenarios

This section lists a number of scenarios showing the deserved behaviour of the system based on the functional requirements from section 5.1. Scenarios are concrete descriptions, depicting how a typical user handles the system, and in what way the system is expected to react. They are hence instances of use cases [BD03].

A special class of scenarios are **boundary scenarios**. They describe special scenarios of shutdown and installation, or in our case system extension.

6.1.1. Regular Scenarios

The first scenario describes the moment when a user is supported in link creation.

Name of Scenario	Creating a Single Link with Link Recommendation
Actors	alice:User
Flow of Events	1. Alice, a UNICASE user, is working on the element “Write tests for our Ultimate Problem Solving Function”. She wants to link this action item with the appropriate functional requirement. 2. Alice navigates to the “Add functional requirement” section of the action item. 3. She spots the requirement “Ultimate Problem Solving Function”, which is displayed on the first place and preselected. 4. She hits the enter-key, thus accepting the suggestion. 5. When navigating back to the work item she finds the item linked.

Table 6.1.: Scenario: Creating a Single Link with Link Recommendation

The second scenario shows how a user can get suggestions for all references of a certain model element.

Name of Scenario	Link Recovery for a Model Element
Actors	bob:User
Flow of Events	1. Bob, a UNICASE user, is changing the textual description of the action item “Proof that NP does not equal P.”. Bob wants that this work item should be linked to as many related elements as possible. 2. Bob selects the function to recommend links for this model element. 3. Bob selects the correct model links to be created. 4. When navigating back to the work item he finds that all selected links are created.

Table 6.2.: Scenario: Link Recovery for a Model Element

The third scenario depicts a recovery of links *after-the-fact*. In this scenario links for the entire project are created.

Name of Scenario	Link Recovery
Actors	charlie:User
Flow of Events	1. Charlie has a look over the project he manages using UNICASE. He realizes that though traceability is very important, his team did let him down regarding this part. At this point he wants to create links between tests and functional requirements. 2. Charlie uses the Link Recovery function for this project. When asked for the elements he wants to trace, he selects tests to functional requirements and the appropriate reference between these elements. 3. Charlie selects the correct links from a list.

Table 6.3.: Scenario: Link Recovery

6.1.2. Boundary Scenarios

This scenario is a boundary scenario. It shows the functionality of how a user can extend the system.

Name of Scenario	Extension
Actors	dave:Developer
Flow of Events	1. Dave is a developer using UNICASE. His projects are mostly documented in a language mostly spoken by the indigenous people of the Christmas Islands. For this language common approaches of Information Retrieval (like VSM) do not work properly and hence recommendation performs quite badly. Thus he wants to extend the recommendation system with his own strategy. 2. Dave writes a recommendation and extends the existing system with his implementation. 3. All suggestions are now created using his home-brew algorithm.

Table 6.4.: Scenario: Extension

6.2. Use Cases

A use case is an abstraction of one or more scenarios [BD03]. According to this definition we abstract the above mentioned scenarios to the following use cases. Furthermore, concrete information to each single use case is given in the subsections underneath.

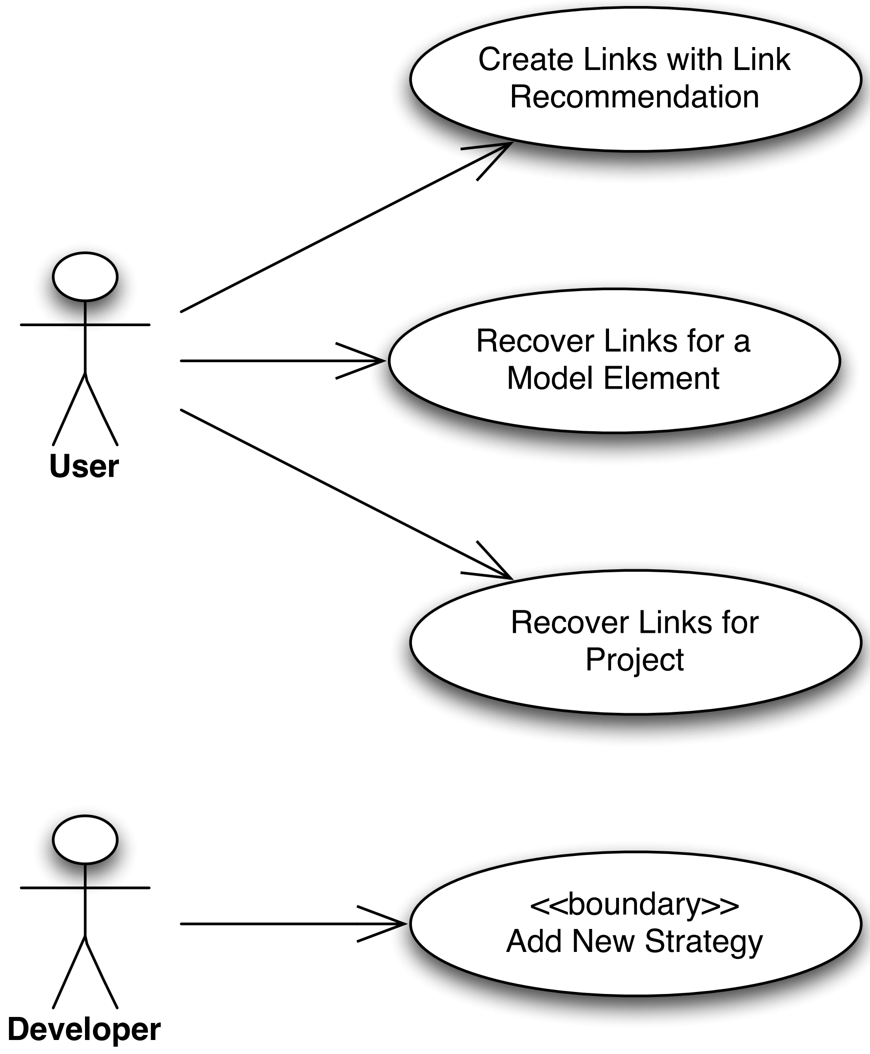

Create Links with Link Recommendation
Recover Links for a Model Element
Recover Links for Project
User
<<boundary>> Add New Strategy
Developer

Figure 6.1.: The System's Use Cases, UML use case diagram

6.2.1. Use Case: Create a Single Link with Link Recommendation

The scenario from table 6.1 can be abstracted to the following use case. Using Abbott's Technique [Abb83] we can furthermore identify the different objects contained in the system.

Use case *Actors*	**Create a Single Link with Link Recommendation** **User** **System**
Flow of Events	1. *User* activates the `LinkButton` in a `model element's editor`. 2. *System* ranks all for this field possible `Elements` and shows all elements in a `SuggestedElementsView`, more probable elements on top, marked with a metric showing some kind of ranking. The most probable element is preselected. All other appropriate items are shown in alphabetical order underneath. 3. *User* either confirms the preselected element by hitting the `enter key`, or selects his preferred element and presses the `accept button` or cancels by clicking the `close button`. 4. *System* creates the desired link (if any), closes the dialog and goes back to the model element's editor.
Entry condition	• The model element's editor is opened.
Exit condition	• The links are created are desired. • The `SuggestedElementsView` is closed.
Quality requirements	• Performance and quality of rankings. • Extensibility: Developers can add their own strategies. • Usability: A right suggestion can be accepted with just one single key press. • Usability: No explanation is needed for a user to use this feature.
Exceptions	• There are no elements available for linking.

Table 6.5.: Use case: Create a Single Link with Link Recommendation

From this use case we can identify the following participating analysis objects:

- `LinkButton`: The button in the `ModelElementEditorView`, which starts a dialog asking what element to connect with this `ModelElement`.

- `ModelElement`: The basic artifact.
- `TraceabilityLink`: A link between two `ModelElement`s.
- `ModelElementEditorView`: A view showing descriptions and offering functions for a `ModelElement`.
- `SuggestedElementsView`: A view showing all possible elements to connect with this element. It offers a possibility to show some items on top of other items.
- `CloseButton`: This button is located on the `SuggestedElementsView`, it closes the window.
- `AcceptButton`: A button accepting the selection, located on the `SuggestedElementsView`.
- `EnterKey`: The key on the keyboard.
- `LinkRecommendationController`: Controls the data flow in this use case.

6.2.2. Use Case: Recover Links for a Model Element

The second use cases derives from the scenario shown in table 6.2.

Use case	**Recover Links for a Model Element**
Actors	**User** **System**
Flow of Events	1. *User* clicks on a `button` at the top of the `model element's editor`. 2. *System* checks for elements of the model not yet set, if it can find a very probable connection. In case the controller is able to find one of those, it adds the elements, but including some kind of `marker` so that the user gets notifications about the system's selection. 3. *User* accepts or declines the system's selection for each of the elements clicking the `accept` or the `cancel button`. 4. *System* saves the suggestion in case the user accepts the recommendation. Otherwise it deletes the suggestion.
Entry condition	• The model element's editor is opened.
Exit condition	• The links are created as desired.
Quality requirements	• Performance: High precision. • Extensibility: Developers can add their own strategies. • Usability: A right suggestion can be accepted with just one single key press.
Exceptions	• There are no elements available for linking.

Table 6.6.: Use case: Recover Links for a Model Element

We identify the following participating analysis objects:

- `ModelElement`
- `TraceabilityLink`
- `ModelElementEditorView`
- `Marker`: A marker as an eye catcher.
- `CancelSuggestionButton`: A button to remove the suggestion.

- `AcceptSuggestionButton`: A button to accept the suggestion.
- `SingleLinkRecoveryController`: Controls the data flow in this use case.

6.2.3. Use Case: Recover Links for the Project

The third use cases derives from the scenario shown in table 6.3.

Use case	**Recover Links for the Project**
Actors	**User** **System**
Flow of Events	1. *User* opens a `project wide recovery view` by selecting it in the appropriate menu of the tool. 1. *User* selects which elements / leafs he wants to analyse and which links he wants to find. 2. *System* analyses where it can search for links and which links are candidates and lists those with a high probability in the view offering an opportunity to accept or decline them. 3. *User* accepts or refuses suggested links by clicking an `accept` or `decline button`. 4. *System* saves the decision.
Entry condition	• none
Exit condition	• The links are created as desired.
Quality requirements	• Performance: High precision. • Extensibility: Developers can add their own strategies. • Usability: A suggestion can be accepted or declined with just one single key press.
Exceptions	• There are no elements available for linking.

Table 6.7.: Use case: Recover Links for the Project

We can identify the following analysis objects:

- `ProjectWideRecoveryView`: A view offering possibilities for selection of element types and reference types. Furthermore suggestions can be placed, accepted and declined.
- `ModelElement`

- `TraceabilityLink`
- `AcceptSuggestionButton`
- `DeclineSuggestionButton`
- `ProjectLinkRecoveryController`: Controls the data flow in this use case.

6.2.4. Boundary Use Case: Add New Strategy

This use case is derived from the boundary use case described in table 6.4.

Use case	**Add New Strategy**
Actors	**Developer** **System**
Flow of Events	1. *Developer* writes a `recommendation strategy` for the system. 2. *Developer* extends the system with his strategy. 3. *System* uses the developer-specified strategy for suggestions.
Entry condition	• none
Exit condition	• System uses developer-specified strategy for suggestions.
Quality requirements	• Implementation: The developer programs in java using the eclipse-framework. • Interface: The developer uses the provided interfaces.
Exceptions	• More than one strategy is registered.

Table 6.8.: Use case: Add New Strategy

Participating analysis objects:

- `RecommendationStrategy`: An abstract object describing a way of finding probabilities for traceability links.
- `RecommendationManager`: Manages the number of strategies in use and selects, which strategy is used for which recommendation.
- `ExtensionView`: Shows existing extensions and offers functionality for changing them.
- `ExtensionController`: Controls the data flow in this use case.

The recommendation strategy is organised in a strategy pattern fashion (see figure 6.2). Thus the concrete implementation of the strategy does not affect the implementation of the controllers or their corresponding views. Moreover, the recommendation manager can change and decide which strategy to use during run time. It will determine this depending on the policy, which is the link type.

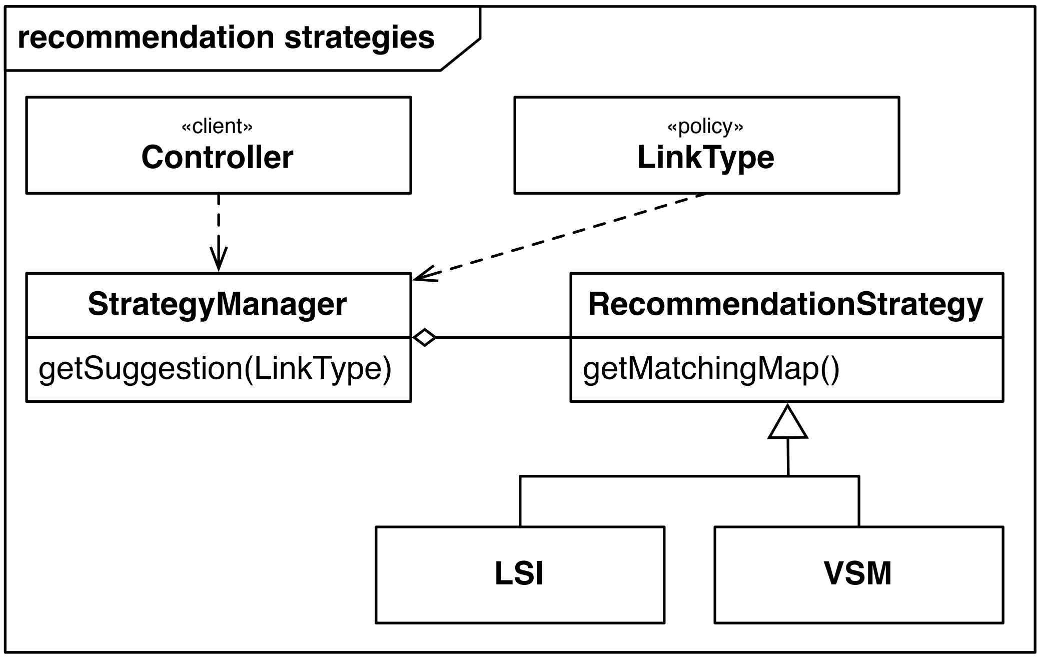


Figure 6.2.: The Strategy Pattern, UML class diagram

6.3. Analysis Object Model

This section will differentiate the participating analysis objects identified in the last section into entity, boundary and control objects.

Use Case	Object-Type	Objects
Creating a Single Link with Link Recommendation	Entity Objects	`ModelElement, TraceabilityLink`
	Boundary Objects	`ModelElementEditorView, SuggestedElementsView, LinkButton, AcceptButton, CloseButton, EnterKey`
	Control Objects	`LinkRecommendationController`
Recover Links for a Model Element	Entity Objects	`ModelElement`, `TraceabilityLink`
	Boundary Objects	`ModelElementEditorView`, `Marker`, `CloseButton`, `AcceptButton`
	Control Objects	`SingleLinkRecoveryController`
Recover Links for the Project	Entity Objects	`ModelElement, TraceabilityLink`
	Boundary Objects	`ProjectWideRecoveryView`, `AcceptButton`, `DeclineButton`
	Control Objects	`ProjectLinkRecoveryController`
Add New Strategy	Entity Objects	`RecommendationStrategy`
	Boundary Objects	`ExtensionView`
	Control Objects	`RecommendationManager, ExtensionController`

Table 6.9.: Entity, Boundary & Control Objects

6.4. Analysis Dynamic Model

This section describes the interaction within the use cases explained in the last section. It explains how the objects operate with each other. As all use cases are similar in their procedure, this section will only show the process behind the use case *Recover Links for the project* (see section 6.2.3).

The sequence diagram (see figure 6.3) shows the interaction between the identified analysis objects from the last section. First the user opens the recovery view. The user selects element and link types. Afterwards the recommendation manager calculates the suggestions, which are displayed to the user. Afterwards the user can accept some of the links suggested. The links a user declines are abandoned, this fairly easy procedure is not displayed in the diagram for space reasons.

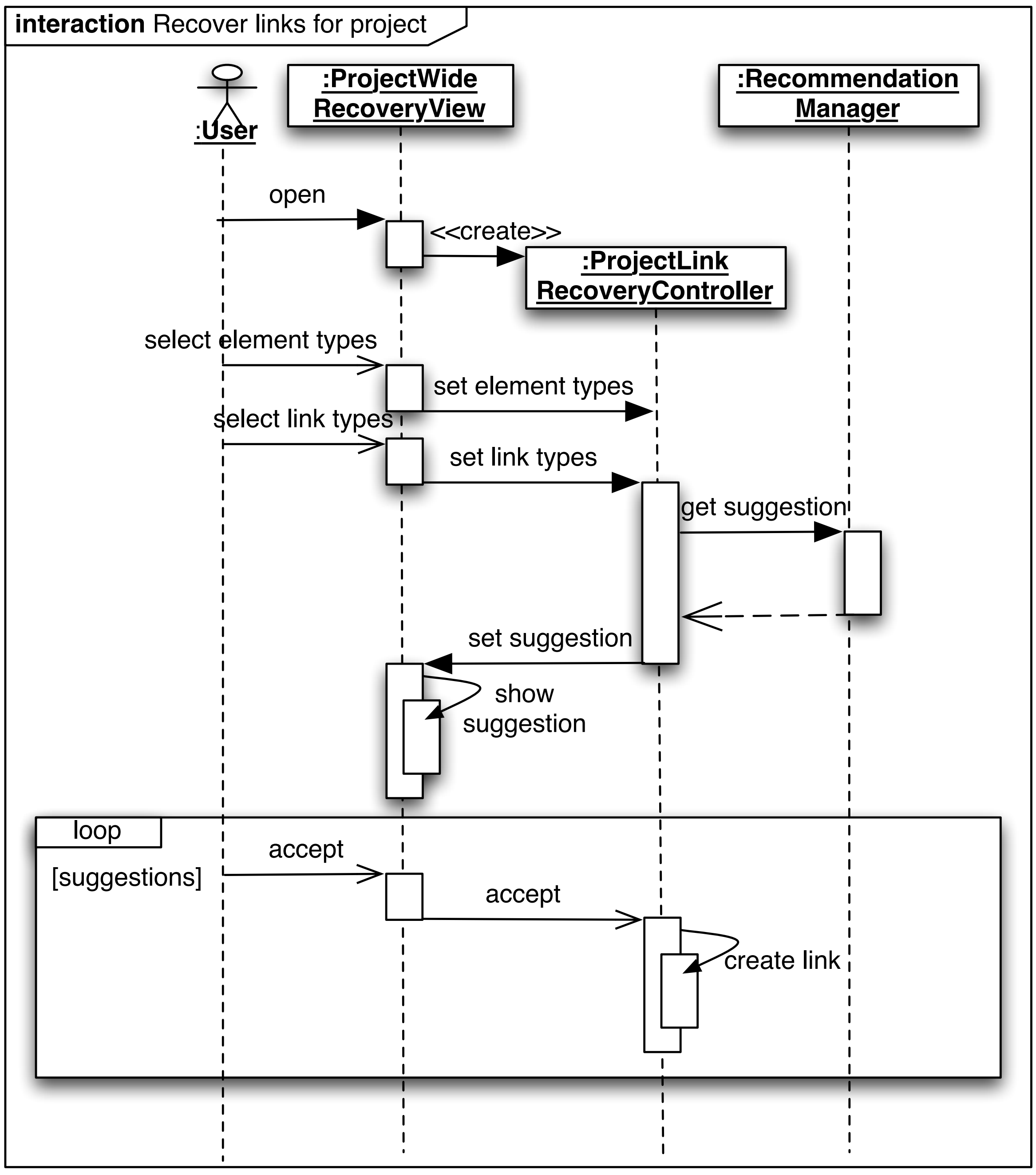


Figure 6.3.: Recover Links for the Project, UML sequence diagram

Part III.

Prototype

7. System Design

≫ *Systems seem, like certain worms, to be formed by a kind of generatio aequivoca - by the mere confluence of conceptions, and to gain completeness only with the progress of time.* ≪

Immanuel Kant, The Critique of Pure Reason

This chapter describes and balances design goals and gives insight into the design of the system in its entirety.

7.1. Design Goals

This section describes certain design goals, according to [BD03] and resulting from section 5.2. The section furthermore balances the design goals and presents their impact on the resulting system.

7.1.1. Design Goals

- *Performance criteria:* The best algorithms shall be used for recommendation. This needs to be evaluated.
- *Cost criteria:* The system must not involve any additional cost to UNICASE. Furthermore the development is limited due to the deadline of this thesis.
- *Maintenance criteria:* The System must be *extensible* and *modifiable*. As techniques change and better algorithms are likely to be developed over the next years, there might be need for change of strategies in future. Change of algorithms in order to improve strategies for a special project is also important to improve the *adaptability*. Furthermore, the system has to comply with the UNICASE code standards for *readability* reasons. Additionally, the system needs to be loosely coupled from the UNICASE project itself, to enable the possibility of delivering the project without link recommendation. Lastly, the recommendation needs to be accessible from the entire UNICASE system, may it be the user interface, the versioning system or the server side service.
- *End user criteria:* The system's main function is to support the user in order to make his work flow faster. Hence, *usefulness* is the major measurement. The user's average work flow needs to accelerate. In order to establish this *usability* requires that the user needs no or hardly any time to understand how to use the framework.

7.1.2. Trade-off of Design-Goals

The above listed design goals obviously conflict at certain aspects.

Performance

The performance includes two different aspects: On the one side, the system needs to react in a certain amount of time, so that the user is not interrupted in his work; on the other side the system's recommendation needs to have a certain quality, so that the results are useful. Usually these are contradicting requirements as better results require more computation. We need to keep this trade-off in mind when deciding which strategies to use. It limits us in the sense, that we can not use algorithms too complex to calculate when time matters.

Still, time does not matter as strongly for the link recovery use cases. Here, if challenged, we should decide on the quality of results, whereas for the link recommendation use case we do not want the user to stop his work and wait for the machine to calculate the results; hence, in this use case we should decide to take the better scaling algorithm.

Precision versus Recall

Obviously both precision and recall are anti proportional to each other through the selection threshold: If the system decides to suggest all values with a probability above a high threshold, the chance that all offered links are correct, is quite high. This is represented by a high precision value. On the contrary, the recall value is going to decrease through the low number of suggested links.

The same refers to the option to lower the threshold: If the system suggests all links above a low threshold, the recall is going to rise, but the precision value is going to fall. This comes from the natural fact that the more elements are selected, the higher is the probability that all desired elements are chosen.

So the question is now where to set the threshold. To answer this question we have to think about the severity of problems we might get from making mistakes. In probability theory this incorrect guessing is sized in the number of false positives and negatives. We identify the incorrectly suggested links as false positives and the incorrectly not suggested links as false negatives.

In addition we have correctly selected links. The elements which are properly guessed to be relevant, in this case the correctly suggested links, are called true positives. The elements which are properly guessed to be not relevant, in this case the links correctly identified to be not correct, are called true negatives.

The damage of a false positive can be measured in the time it takes the user to correct it, whereas the benefit of a true positive can be measured as the time a user saves in comparison to the time the user would have needed if he had chosen the link manually.

The damage of a false negative can be the time it takes to find it manually instead of selecting or accepting the suggestion. The benefit of a true negative is just that it does not cause any damage as being a false positive. The aim should always be to minimise the total value of the total damage.

Bringing back the precision and recall into this calculation:

- The higher the precision, the higher the number of false negatives, but the lower the number of true positives.
- The higher the recall, the higher the number of true positives, but also the higher the number of false positives.

As a conclusion to the presented phenomena the system has to carefully adjust the threshold from case to case. If a damage of a false positive is quite high according to the benefit of a true positive, we should set the threshold quite high, thus getting a high precision and hereby a low number of false positives. This leads, as desired, to a low amount of damage through false positives.

This is important for example in the link recommendation use case, as it does not really matter if the system misses out a suggestion as long as the other suggestions are correct. But if the damage of false positives is relatively low in comparison to the benefit of a true positive, we should lower the threshold as this leads to a higher recall, which leads to a higher number of true positives. This is important in the link recovery use case, as a user has a high benefit from directly selecting a link in comparison to the damage it does when an inadequate link is suggested.

As this section tried to point out, the right choice of strategies is essential for the success of the system. Therefore an evaluation is inevitable. We will conduct this evaluation in chapter 10.

Extensibility versus Performance

Another issue, which is not as obvious, is the trade-off extensibility versus performance. In order to enable extensibility, all recommendation strategies need to be reduced to the least common denominator. To keep the extension simple, a simple interface needs to be used. This, however, limits the strategies in their possibilities, as new strategies need to stick to the same pattern.

7.1.3. Impact on Design

In order to establish the loose coupling wanted by the design goals, dependencies on the recommendation framework must be minimised. Therefore a manager is created as the centralised point of attaching and detaching strategies. This manager must be localised within the centre part of UNICASE; thus giving the opportunity to access recommendation at every part of the system. This does not violate the need for delivering UNICASE without link recommendation: If delivery without recommendation is wanted, no strategies are plugged in. This manager is furthermore the centre for extension.

For performance reasons, an evaluation will be done to find out the best strategy. For usability reasons complex systems with user response et cetera were abandoned and simple solutions preferred.

7.2. System Architecture

Different subsystems can be distinguished in the framework:

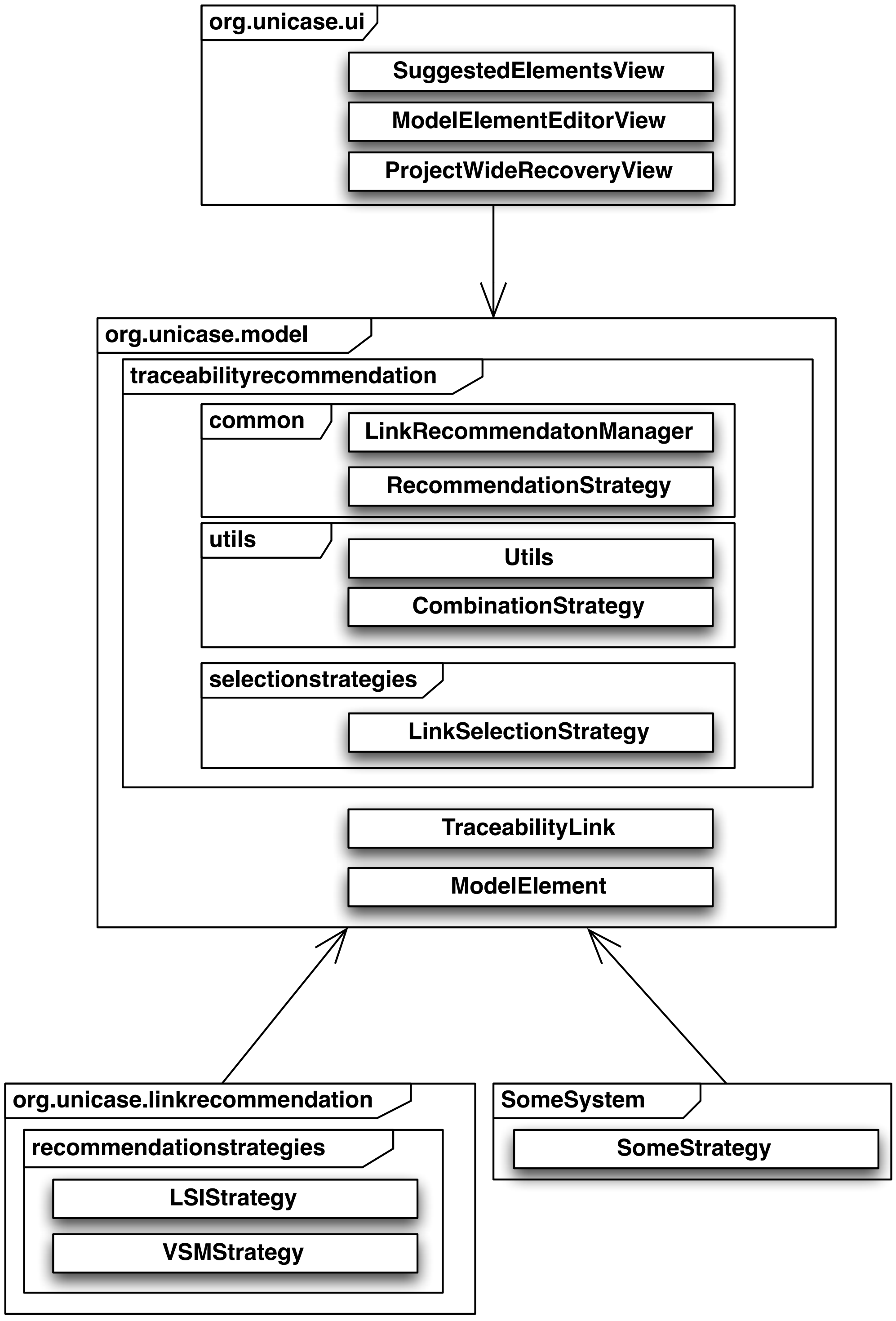


Figure 7.1.: The Framework's Architecture, UML class diagram

In detail, the tasks of these subsystems are:

Name	**Purpose**
`org.unicase.ui`	In this subsystem all views are contained. It depends on `org.unicase.model` for mainly two reasons: the link recommendation and the different model elements.
`org.unicase.model`	The main subsystem of UNICASE. In this subsystem are most classes, which can be accessed from various parts of UNICASE.
`org.unicase.model .traceabilityrecommendation.common`	This is the central control part of the framework. In here, strategies are registered and suggestions are made.
`org.unicase.model .traceabilityrecommendation.utils`	This subsystem includes utilities and features used by different strategies.
`org.unicase.model .traceabilityrecommendation .selectionstrategies`	This subsystem includes the interface as well as the implementations of link selection strategies as proposed in section 2.5.
`org.unicase.linkrecommendation .recommendationstrategies`	This subsystem contains the various implementations proposed by this thesis.
`SomeSystem`	This is an example for an extended strategy by a third-party-developer.

Table 7.1.: The Subsystems

This system is designed as follows: The main part of the entire system is the subsystem `traceabilityrecommendation.common`. It is located in the UNICASE subsystem `org.unicase.model` (this subsystem also contains important UNICASE objects such as the `ModelElement`). All queries for link recommendation go to this subsystem. It needs to be placed in this subsystem, so that it can be used at every point in the UNICASE project. This subsystem can not only be used by the user interfaces, but also by other parts of the UNICASE system, for example by an automated procedure on the UNICASE repository server.

In order to avoid code repetition (according to the DRY principle *Don't Repeat Yourself*), a centralised utility subsystem `traceabilityrecommendation.utils` is offering predesigned classes for combination of strategies and methods for counting and stemming words.

Furthermore selection strategies and the according interface are contained in another subsystem `traceabilityrecommendation.selectionstrategies`. This needs to be accessible from the recommendation manager, thus it is placed in the system `org.unicase.model` as well.

The various views identified in section 6.3 can request suggestions from the recommendation subsystem. But as most parts of the system do not need access to these views (for example all server side parts, the so called EMFStore), they are extracted from `org.unicase.model` and localised with other user interface classes in the subsystem `org.unicase.ui`. As this subsystem needs admittance to `org.unicase.model`, there is a dependency from the `org.unicase.ui` to `org.unicase.model`.

The strategies themselves are only extensions to the main system. This is decoupled, so that all strategies are optional and can be exchanged at leisure. Some strategies were created for this thesis. They can be found in the subsystem `org.unicase.linkrecommendation`. For readability reasons most of the strategies were omitted in the diagram above, they are all contained in the subsystem `org.unicase.linkrecommendation`. To show how an extending system `SomeSystem` fits into the architecture, this was added into the diagram. All extensions need access to the `RecommendationStrategy` and create an extension for the plugin provided by `org.unicase.model`, hence all extensions depend on the subsystem `org.unicase.model`.

8. Object Design

≫ *Is mios' an t-eagal na 'n cogadh.*
(Carelessness is worse than theft) ≪

Old Gaelic Proverb

This chapter maps the analysis objects from section 6.3 onto real objects. It is organised according to the identified subsystems from the previous chapter.

1. `org.unicase.model`: In this section the recommendation manager and all attached classes are explained.
2. `org.unicase.linkrecommendation`: This section shows the interrelation and structure of recommendation strategies.
3. `org.unicase.ui`: In this section this framework's user interfaces are explained.

8.1. org.unicase.model

This subsystem contains the implementation of the class `ModelElement`, as well as the recommendation manager and the interface for recommendation strategies. This section will additionally explain the selection strategies, which are implemented in `org.unicase.model` as well. Lastly, this subsystem includes the utility classes.

8.1.1. The ModelElement and Links

The basic class for all ModelElements is located in the package `org.unicase.model`. It is provided by the UNICASE project. The traceability links are not real entities. Linked

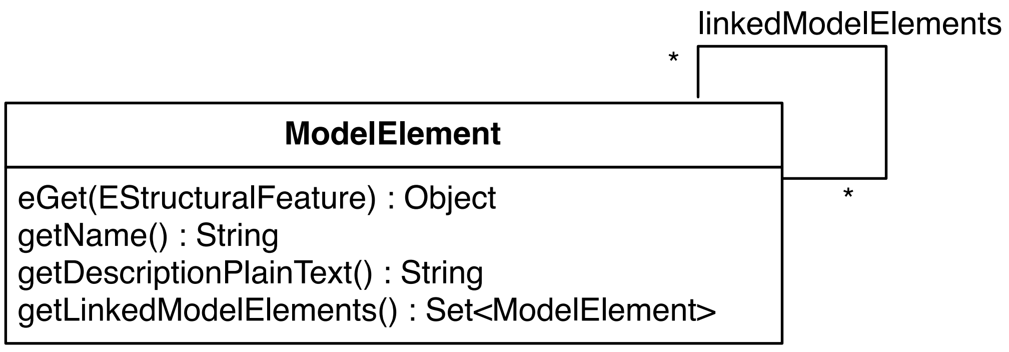

linkedModelElements
*
*
ModelElement
eGet(EStructuralFeature) : Object
getName() : String
getDescriptionPlainText() : String
getLinkedModelElements() : Set<ModelElement>

Figure 8.1.: The Model Element, UML class diagram

ModelElements can be retrieved by the link type (an `EReference` object) via the method `eGet`. The result is either an `EList` of `ModelElements` or a single `ModelElement`. Linked model elements can also be retrieved by the `getLinkedModelElements` method (see figure 8.1). The methods `getName` and `getDescriptionPlainText` provide the textual content of a model element.

8.1.2. The RecommendationManager

Interface RecommendationStrategy

All strategies need to implement the interface `RecommendationStrategy` (see figure 8.2). It contains two methods:

- `getName() : String`
- `getMatchingMap(ModelElement, Collection<ModelElement>)`
  `: Map<ModelElement, Double>`

The first method, `getName`, should return the name of this strategy. It is used for output purposes.

The second method, `getMatchingMap`, includes the actual calculation. The method should return an object including mappings for a number of elements (not necessarily for all). The base element and a list of candidates is provided for calculation. The usual modus operandi is to iterate over all candidates and somehow calculate a number for each. For standardisation reasons the results should be normalised to values between zero and one. Lastly the developer returns a map with pairs of candidate and associated value. In order to be able to reuse the implementations of `Map` provided by the java API the type `Double` is used instead of the primitive data type `double`.

RecommendationManager

The recommendation manager (see figure 8.2) is the centre of the entire project. Whenever a subsystem in UNICASE needs recommendation, it calls on of the recommendation manager's methods. In order to prevent that the recommendation manager checks for extensions every time a recommendation needs to be made, it is designed as a *singleton*. The instance can be retrieved with the `getInstance` method. The recommendation manager uses both selection and recommendation strategies in a *strategy pattern* fashion. It combines context and policy, deciding which strategy to use based on the parameters. To get a strategy for a certain link type, the developer can use the method `getRecommendationStrategy`. The developer needs to provide the base element and the targeted reference as parameters. The method `getMatchMap` creates an complete suggestion. The parameters are approximately the same to the `getRecommendationStrategy` method, additionally the candidates and the selection strategy to be used (see section 8.1.3) is needed. The method returns a map giving a `Double` value for a number of model elements. If `null` is set as selection strategy, no selection is made and only the result of the recommendation strategy's `getMatchingMap` method is forwarded.

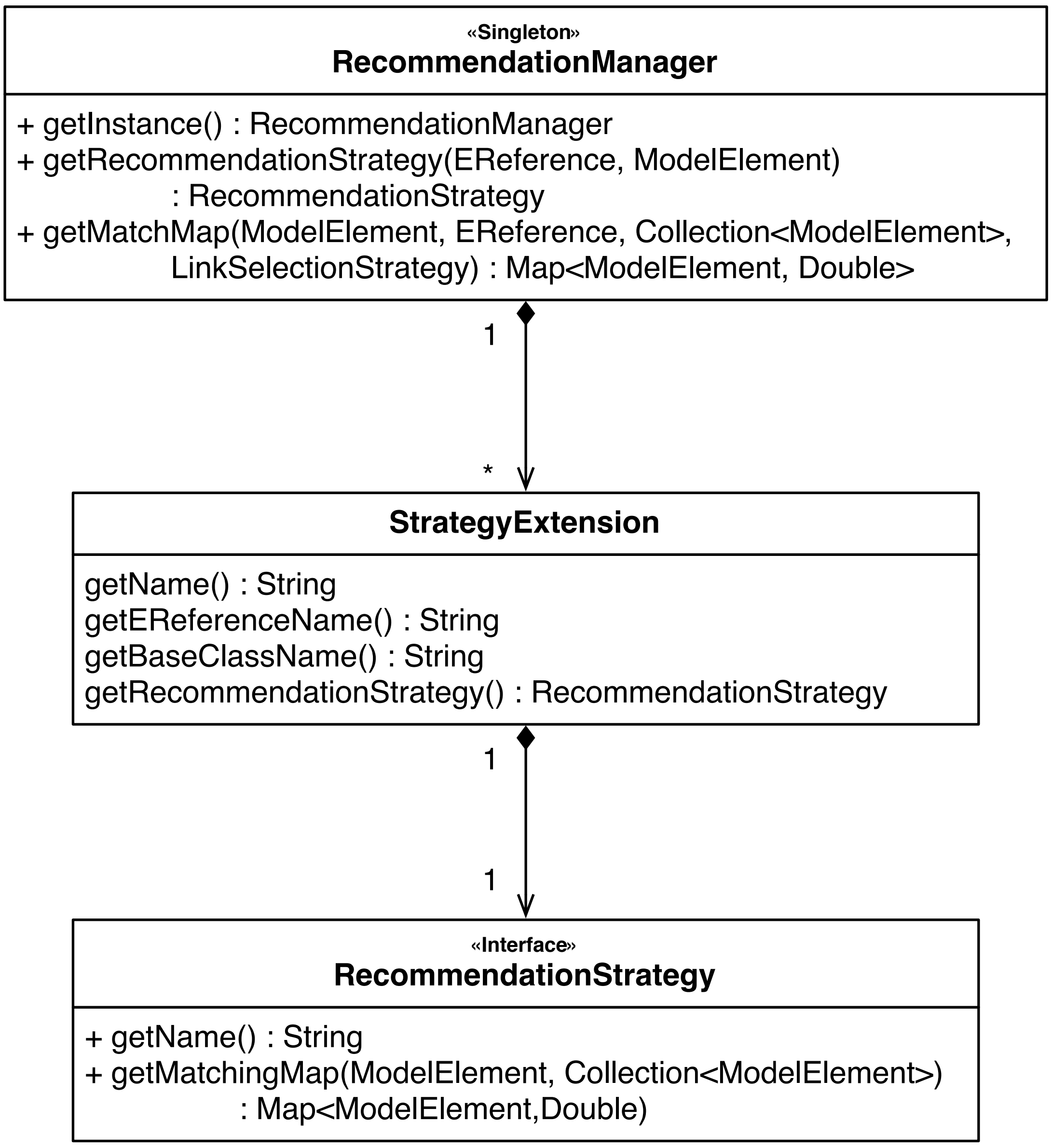

«Singleton»
RecommendationManager
+ getInstance() : RecommendationManager
+ getRecommendationStrategy(EReference, ModelElement) : RecommendationStrategy
+ getMatchMap(ModelElement, EReference, Collection<ModelElement>, LinkSelectionStrategy) : Map<ModelElement, Double>
1
*
StrategyExtension
getName() : String
getEReferenceName() : String
getBaseClassName() : String
getRecommendationStrategy() : RecommendationStrategy
1
1
«Interface»
RecommendationStrategy
+ getName() : String
+ getMatchingMap(ModelElement, Collection<ModelElement>) : Map<ModelElement,Double)

Figure 8.2.: The Recommendation Manager, UML class diagram

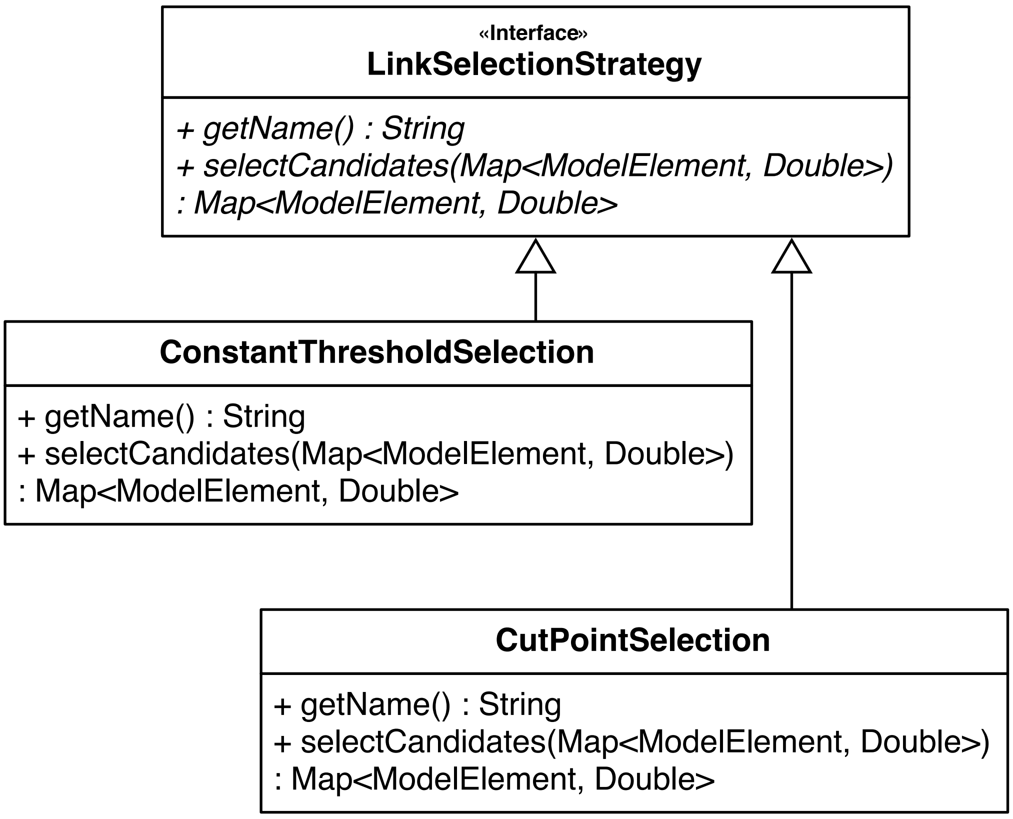

«Interface»
LinkSelectionStrategy
+ getName() : String
+ selectCandidates(Map<ModelElement, Double>)
: Map<ModelElement, Double>
ConstantThresholdSelection
+ getName() : String
+ selectCandidates(Map<ModelElement, Double>)
: Map<ModelElement, Double>
CutPointSelection
+ getName() : String
+ selectCandidates(Map<ModelElement, Double>)
: Map<ModelElement, Double>

Figure 8.3.: The Selection Strategies, UML class diagram

8.1.3. The Selection Strategies

The selection strategies are implemented in the package `org.unicase.model.util .traceabilityrecommendation.selectionstrategies` (see figure 8.3).

The LinkSelectionStrategy Interface

The selection strategy interface contains two methods:

- `getName() : String`
- `selectCandidates(Map<ModelElement, Double>)`
  `: Map<ModelElement, Double>`

The first method `getName` again only returns a name of the selection strategy, in order to be able to print it out for evaluation or debug.

The second method, `selectCandidates` filters the result of a link recommendation. It takes a map from `ModelElements` to `Double` and usually removes a number of elements. The result represents those elements the system would suggest to the user.

The Selection Strategies

This framework provides two subclasses for the interface `LinkSelectionStrategy`. First, the `ConstantThresholdSelection`: It implements the method `selectCandidates` using a

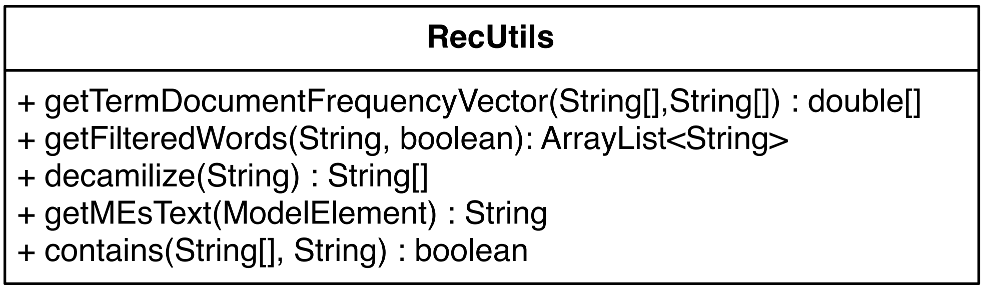

RecUtils
+ getTermDocumentFrequencyVector(String[],String[]) : double[]
+ getFilteredWords(String, boolean): ArrayList<String>
+ decamilize(String) : String[]
+ getMEsText(ModelElement) : String
+ contains(String[], String) : boolean

Figure 8.4.: The RecUtils, UML class diagram

constant threshold like proposed in 2.5. All model elements with a value under a certain threshold, which is passed in the constructor, are discarded. The second implementation is the `CutPointSelectionStrategy`. This implementation of the `selectCandidates` method sorts the model elements by their value. A parameter, again passed in the constructor, defines the cut point x. The topmost x elements are then passed through, all others are discarded.

8.1.4. The Utilities

This framework provides two classes with utilities for information retrieval work. The first is the class `RecUtils`, which contains methods for word extraction and counting. The second class is `TDFrequencyMatrix`, which is a matrix for special information retrieval purposes.

RecUtils

`RecUtils` (see figure 8.4) contains five different methods for different purposes:

- `getTermDocumentFrequencyVector(String[] dictionary,String[] document) : double[]` This method counts the words of the first parameter in the second parameter, e.g. if the dictionary was {PC, MAC} and the document was {MAC,PC,MAC}, the method would return {1,2} as the first word (PC) appeared once in the document and the second word (MAC) appeared two times.
- `getFilteredWords(String, boolean) : ArrayList<String>` This method extracts the words from the first parameter. The second parameter determines whether stemming is applied or not. The method filters *stop words* (like *the, do* et cetera) and extracts commas, colons etc. (see section 2.4.1 for details).
- `decamilize (String) : String[]` This method cuts CamelCaseWords, the implementation is based on [Mal09], see section 2.4.1.
- `getMEsText(ModelElement) : String` The purpose of this method is extraction of textual content from ModelElements.
- `contains(String[], String) : boolean` This helper for arrays returns true if the first parameter contains the second, false otherwise.

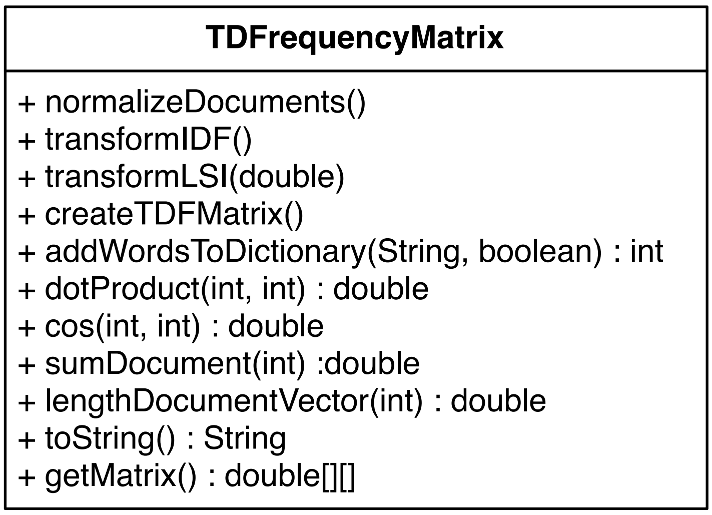

TDFrequencyMatrix
+ normalizeDocuments()
+ transformIDF()
+ transformLSI(double)
+ createTDFMatrix()
+ addWordsToDictionary(String, boolean) : int
+ dotProduct(int, int) : double
+ cos(int, int) : double
+ sumDocument(int) :double
+ lengthDocumentVector(int) : double
+ toString() : String
+ getMatrix() : double[][]

Figure 8.5.: The TDFrequencyMatrix, UML class diagram

TDFrequencyMatrix

The class `TDFrequencyMatrix` (see figure 8.5) is a class symbolising a term to document matrix offering methods for content based information retrieval (see section 2.3). Each object of this class stands for a term to document matrix. The developer first fills the matrix with documents using the method `addWordsToDictionary`, the first parameter is the text, the second parameter determines if stemming is applied. It returns the document's index in the matrix. The matrix is then calculated with the method `createTDFMatrix`, this includes only counting of words. The method `normalizeDocuments` transforms the matrix, so that the sum of each vector (or document) is one. The sum of all elements of a vector can be retrieved with the method `sumDocument` giving the document's index as parameter. The method `lengthDocumentVector` calculates the length of a document's vector (in 2-norm). The class also offers methods for the cosine of two vectors and the dot product, the parameters are always the indices of the documents. Furthermore transformations according to section 2.4 are provided with the methods `transformIDF` and `transformLSI(double)`. For LSI the developer has to pass the k-value as parameter.

8.2. org.unicase.linkrecommendation

We implemented the strategies used in this thesis in `org.unicase.linkrecommendation`, whereas the abstractions are implemented in a subsystem of `org.unicase.model` , so that the entire system, especially the recommendation manager, has access to it. This section shows the strategies and their class hierarchies (see figure 8.6).

8.2.1. Basic Strategies

The content based approaches LSI and VSM are both based on term-to-document matrixes, therefore they both have an `TDFrequencyMatrix`, reusing this code. All approaches are based on the interface `RecommendationStrategy` explained above. The history-based approach

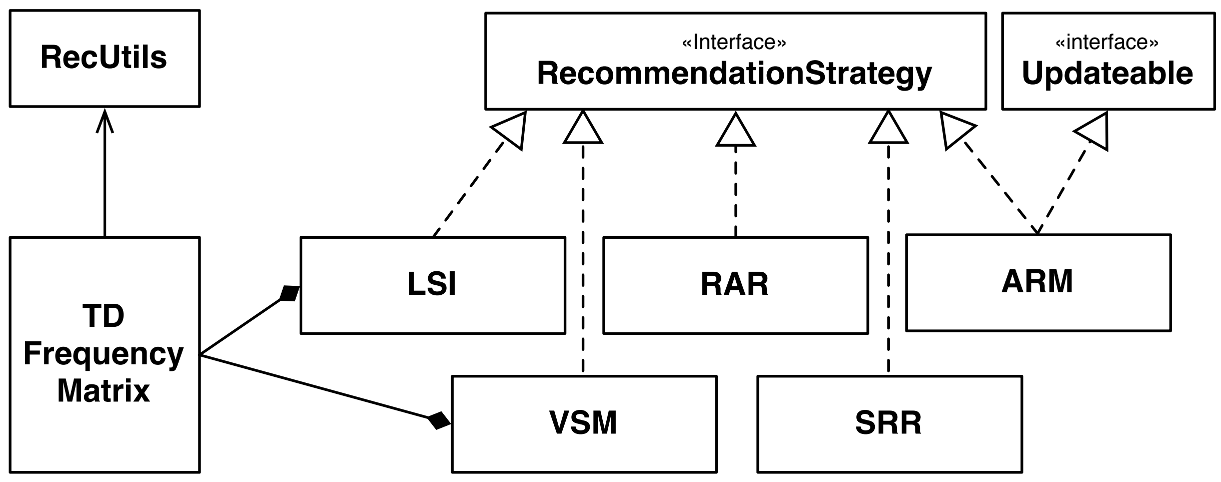


Figure 8.6.: The Strategies, UML class diagram

ARM needs to be handled in a different manner, this will be explained in the next section. All other approaches are implemented in the way explained in the section 2.4.

8.2.2. Updateable Strategies

Training is essential for the history-based approach ARM. To offer this functionality, we created a different type of strategies with the interface `Updateable`. This only contains one method: `updateStrategyData(ChangePackage)`. By creating this interface, the system can check for the strategies used whether it needs to train it, without knowing the single class types. Combinations of strategies need to pass this functionality, this will be detailed in the next section.

8.2.3. Combination Strategies

As one of the requirements was to create possibilities for combination of strategies, another subtype of strategies was created. The abstract class `CombinationStrategy` provides this functionality (see figure 8.7). It is implemented in the core, so that the entire system has access to it.

It offers the possibility to combine two strategies, which are transmitted via the constructor. It contains one abstract method `combine`, which determines how the different approaches are combined. This method is then implemented, according to the considerations from section 2.4, in the subclasses `MaximumCombinationStrategy` and `FactorCombinationStrategy`.

In order to use also updateable strategies (see section 8.2.2), the abstract class implements the interface `Updateable` as well. It implements the required update method by checking if any of the two participating strategies are updateable, and if so, forwarding the incoming update to the participating strategies.

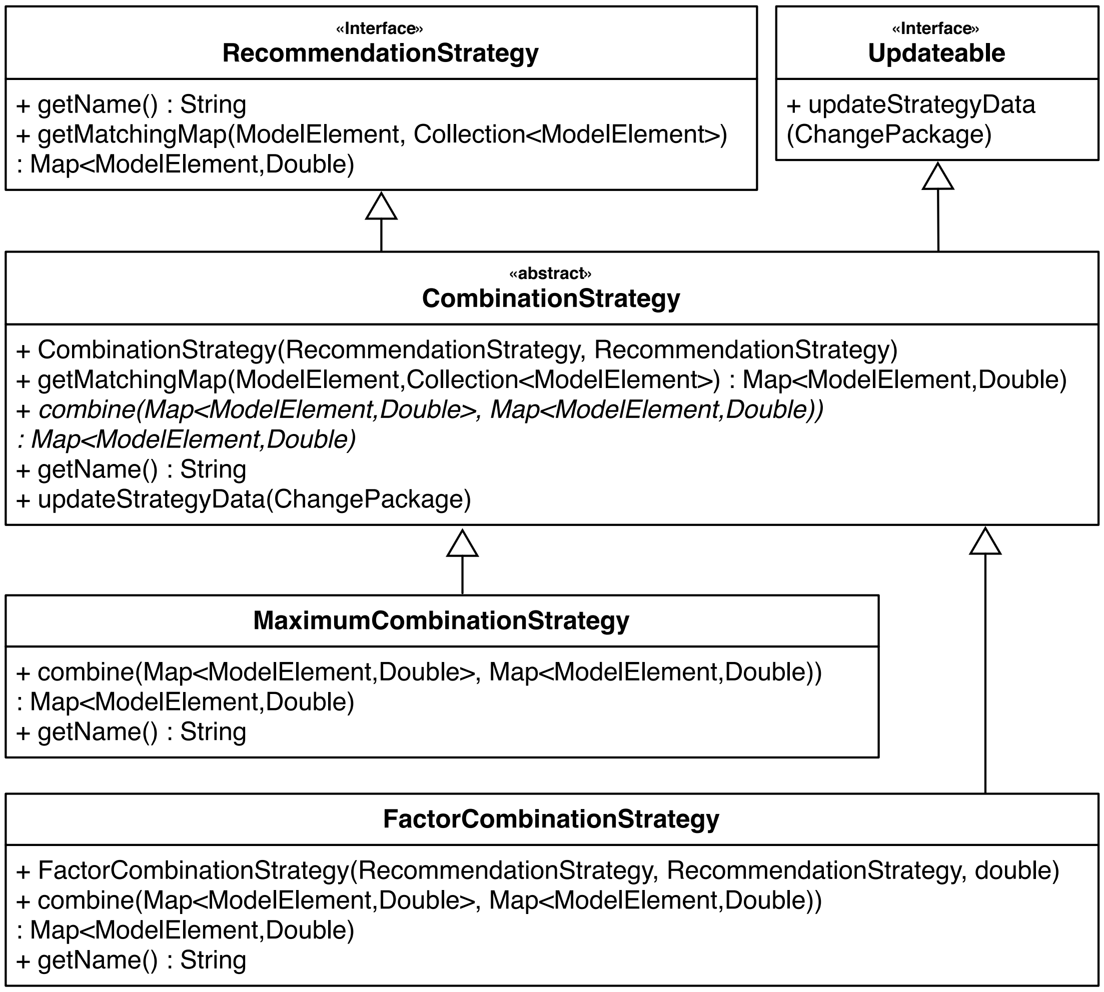

«Interface»
RecommendationStrategy
+ getName() : String
+ getMatchingMap(ModelElement, Collection<ModelElement>)
: Map<ModelElement,Double)
«Interface»
Updateable
+ updateStrategyData
(ChangePackage)
«abstract»
CombinationStrategy
+ CombinationStrategy(RecommendationStrategy, RecommendationStrategy)
+ getMatchingMap(ModelElement,Collection<ModelElement>) : Map<ModelElement,Double)
+ combine(Map<ModelElement,Double>, Map<ModelElement,Double))
: Map<ModelElement,Double)
+ getName() : String
+ updateStrategyData(ChangePackage)
MaximumCombinationStrategy
+ combine(Map<ModelElement,Double>, Map<ModelElement,Double))
: Map<ModelElement,Double)
+ getName() : String
FactorCombinationStrategy
+ FactorCombinationStrategy(RecommendationStrategy, RecommendationStrategy, double)
+ combine(Map<ModelElement,Double>, Map<ModelElement,Double))
: Map<ModelElement,Double)
+ getName() : String

Figure 8.7.: The Combination Strategies, UML class diagram

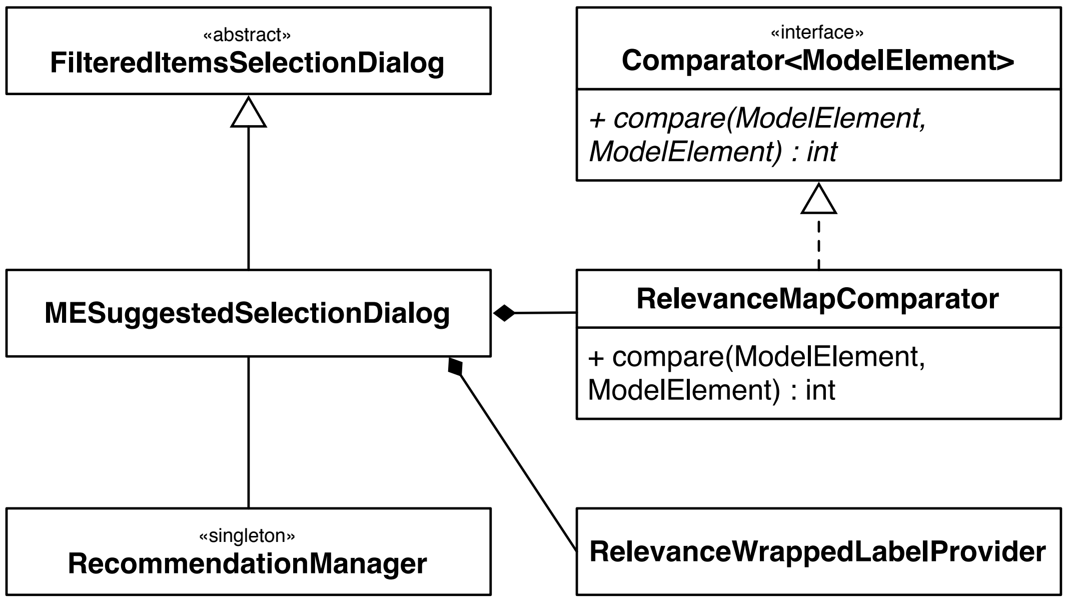

«abstract»
FilteredItemsSelectionDialog
«interface»
Comparator<ModelElement>
+ compare(ModelElement, ModelElement) : int
MESuggestedSelectionDialog
RelevanceMapComparator
+ compare(ModelElement, ModelElement) : int
«singleton»
RecommendationManager
RelevanceWrappedLabelProvider

Figure 8.8.: The MESuggestedSelectionDialog, UML class diagram

8.3. org.unicase.ui

This section describes the user interfaces and their access to the framework. It contains two different views. On the one side, the `MESuggestedSelectionDialog`. This view is designed to show a number of elements for link recommendation, having some elements presented more prominently. On the other side, there is a view for link recovery. This is realised in a wizard, called `RecoveryWizard`.

8.3.1. MESuggestedSelectionDialog

Eclipse offers different dialogs for extension. For this thesis a dialog is needed, which has functionality to select an item out of many and has an influence on the sorting. These constraints are fulfilled by the abstract class `FilteredItemsSelectionDialog` (see figure 8.8). The system sorts the elements using a provided `Comparator`. The comparator `RelevancemapComparator` works as follows: If none of the elements are under the suggested elements, the items are compared alphabetically. If one and only one item is suggested, this one is returned to be higher. If both elements are suggested, the element with a higher probability value is stated to be higher. For these suggestions the dialog refers to the recommendation manager singleton as described in the previous chapters.

Eclipse widgets use so called *LabelProviders* for the display of items. The UNICASE project offers such a class. For our purpose this class was extended, so that not only the name of an element is displayed in the dialog, but also the probability value, if any provided.

To fulfil the requirement that UNICASE can be delivered without this framework, the dialog also works without any recommendation, then only displaying the items in alphabetical order. This dialog replaced the dialog for adding links from the legacy system.

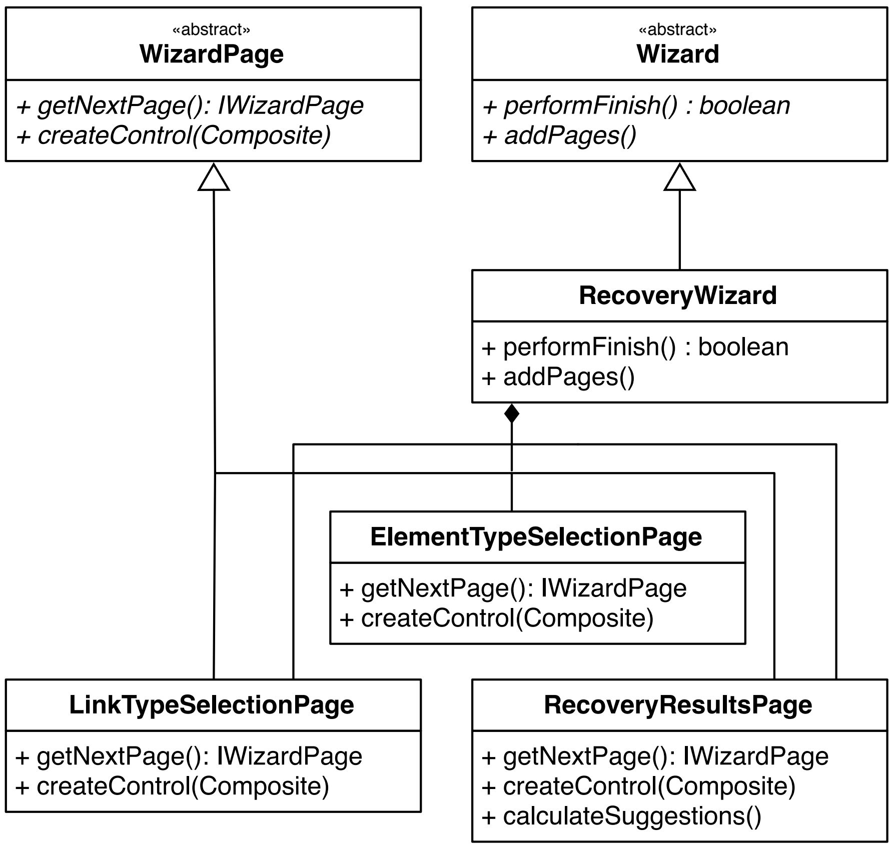

«abstract»
WizardPage
+ getNextPage(): IWizardPage
+ createControl(Composite)
«abstract»
Wizard
+ performFinish() : boolean
+ addPages()
RecoveryWizard
+ performFinish() : boolean
+ addPages()
ElementTypeSelectionPage
+ getNextPage(): IWizardPage
+ createControl(Composite)
LinkTypeSelectionPage
+ getNextPage(): IWizardPage
+ createControl(Composite)
RecoveryResultsPage
+ getNextPage(): IWizardPage
+ createControl(Composite)
+ calculateSuggestions()

Figure 8.9.: The Recovery Wizard, UML class diagram

8.3.2. RecoveryWizard

The second user interface is used for both project wide link recovery and link recovery for single items. It is implemented in the class `RecoveryWizard` (see figure 8.9); a `Wizard` is a feature which eclipse offers for user interaction. It contains several pages, usually derived from the abstract class `WizardPage`. The functionality of the `RecoveryWizard` follows the description displayed in figure 6.3 in the analysis chapter. In our case, we have three distinct page types:

- `ElementTypeSelectionPage:` This page shows all different model element types. It is used for both selection of base and candidate element types. The user can tick the items desired.
- `LinkTypeSelectionPage:` This page lets the user select the designated link types. Furthermore is displayed, for which base types these link types are available.
- `RecoveryResultsPage:` The last page displays the suggestions from the recommendation. These are calculated with the method `calculateSuggestions`. The user can select the favoured suggestions and by hitting the *Finish* button, these links are created. In case of uncertainty, the user may also click on the suggested elements to open the element's detailed editors.

WizardPage

For this thesis two methods from the abstract class `WizardPage` are important: `getNextPage` is called whenever the user clicks the *Next* button and when the page is loaded. The pages from this thesis usually store and forward the user's selection in this method. The second method is `createControl`. This method offers the developer the possibility to add design elements to the wizard page. These are added to the `Composite` given as parameter.

The class `WizardPage` is an implementation from the interface `IWizardPage`.

RecoveryWizard

The class `WizardPage` has two important methods. `performFinish` is called whenever a user presses the finish button. This method checks the selected elements from the `RecoveryResultsPage` and adds the designated links. The second method, `addPages`, creates and initialises the wizard's pages. This method checks if the user wants links for the entire project or only for one element. For the latter only the results page is displayed.

These methods are overwritten from the class `Wizard`. The abstract class `Wizard` again is an implementation of the interface `IWizard`.

9. A Manual for Extending the Link Recommendation Framework

≫ *Example isn't another way to teach, it is the only way to teach* ≪

Albert Einstein, Nobel Price winner

This chapter presents how the framework can be extended using Eclipse extension points. In order to make the reader fully understand how to correctly write and include one's own extension, the extension point system is explained shortly. Afterwards the paper shows an example extension. This is not meant as a deep introduction into the Eclipse extension point technology; the interested reader shall be referred to [Vog09] or [GB03].

9.1. Eclipse Extension Points

Eclipse offers functionality to create extensible software with extension points and extensions. In order to accomplish this, the developer defines an extension point within his plugin. He creates a scheme, including different descriptors, for example a name and an id, but most importantly including a class parameter. The class parameter references the class to be used by the extension point. It is usually implementing an interface or extending a special super class.

The creator of the extension point can check, which extensions for his extension point are available and instantiate the classes, which provide the additional functionality. He does not know in compile time which plugins are provided, but needs to evaluate this during run time.

The developer, who wants to add functionality, adds an extension to his plugin. In this extension he defines the name for his special extension and adds the classes, which are meant to provide the additional functionality.

9.2. An Example

Adding a strategy to the framework just takes two steps:

1. Write a strategy.
2. Create the extension.

These will be explained in detail below.

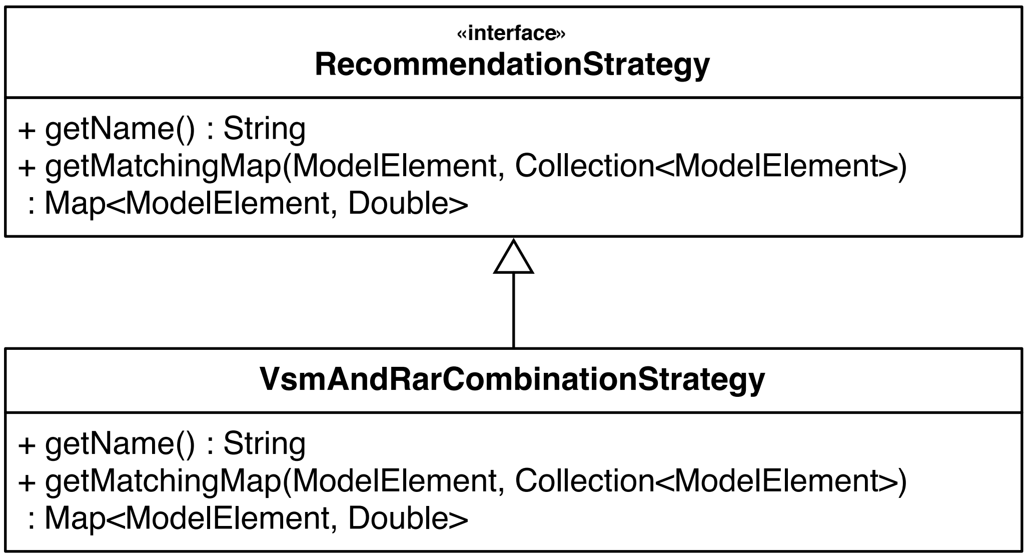

«interface»
RecommendationStrategy
+ getName() : String
+ getMatchingMap(ModelElement, Collection<ModelElement>) : Map<ModelElement, Double>
VsmAndRarCombinationStrategy
+ getName() : String
+ getMatchingMap(ModelElement, Collection<ModelElement>) : Map<ModelElement, Double>

Figure 9.1.: Extension Manual: The Strategy Class, UML class diagram

9.2.1. Writing a Strategy

First of all the developer needs to create the functionality. To achieve this he needs to write a class implementing the interface `RecommendationStrategy` from the package `org.unicase.model.common.traceabilityrecommendation`. This interface requires implementation of only two methods:

- `getName() : String`
- `getMatchingMap(ModelElement, Collection<ModelElement>)`
  `: Map<ModelElement, Double>`

The first method, `getName`, should return the name of this strategy. It is used for output purposes.

The second method, `getMatchingMap` includes the actual calculation. The method should return an object including mappings for a number of elements (not necessarily for all). The base element and a list of candidates are provided for calculation. The usual modus operandi is to iterate over all candidates and somehow calculate a number for each. For standardisation reasons the results should be normalised to values between zero and one. Lastly the developer returns a map with pairs of candidate and associated value.

Our example developer wrote a combination of the vector space model approach and the related assignees approach. The resulting class is shown in figure 9.1.

9.2.2. Creating the Extension

Now, that the class is created, it needs to be “plugged in” into the framework. The developer can define the base type this strategy should be used with and additionally the link type it is designed for. If a strategy is meant to be used for all types of elements, one can enter the class `org.unicase.model.ModelElement`. If a strategy should be used for all kinds of links the developer can enter the keyword ALL.

The following steps need to be executed in this part:

1. First of all the developer needs to make sure, that he does not have any recommendation strategies for this combination of base element type and eReference type already used in a strategy extension.

2. Then he switches to the `plugin.xml` or the `MANIFEST.MF` file and there to the extension point tab.

3. He clicks on `Add`, selects the `org.unicase.model.recommendationstrategy` extension point and accepts it (see figure 9.2).

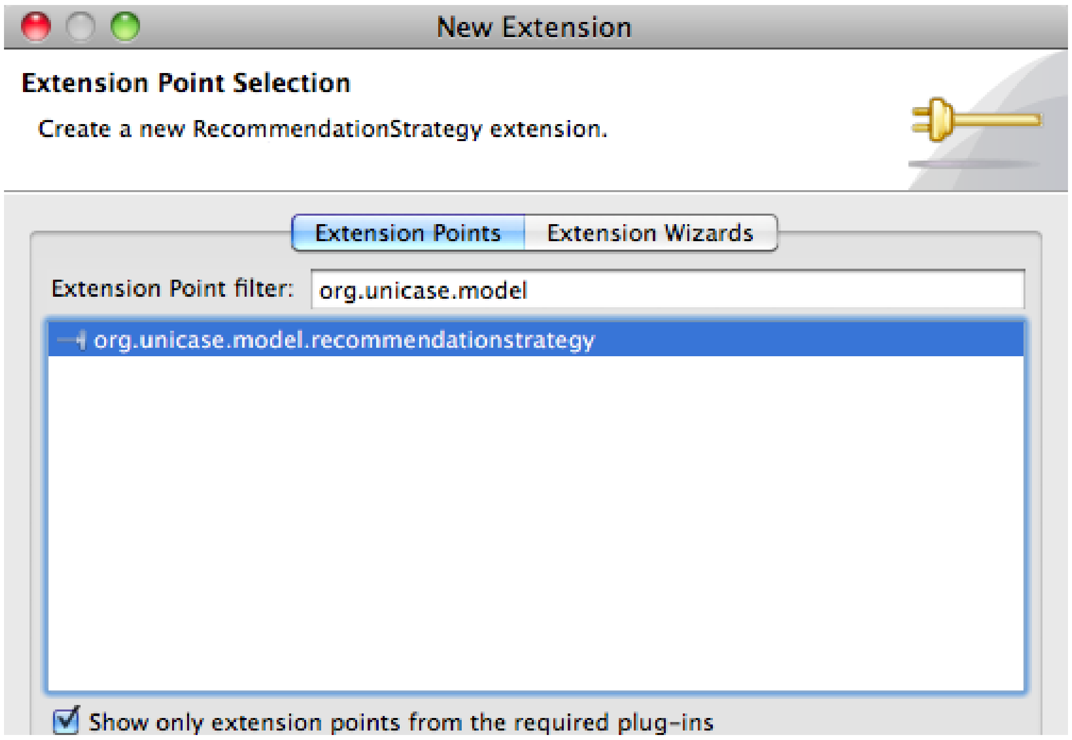
4. He right-clicks on the newly appearing element in the extensions panel and selects new $\rightarrow$ strategy (see figure 9.3)

5. He enters a name in the extension elements details panel of the new strategy (see figure 9.4). In our case, he names it `org.unicase.linkRecommendation.VsmAndRar`.

6. He selects the strategy class by clicking the browse button and searching for the appropriate class (see figure 9.5). For this case, this is the `VsmAndRarCombinedStrategy` class.

7. He enters the reference this strategy shall be used for. If it can be used for all references, ALL can be entered. Otherwise the developer needs to enter the eReferences name. It can be retrieved from the `getName` method of the reference. As the new strategy shall just work on the reference `annotatedModelElement`, this is entered here.

8. Lastly the type of model elements this strategy shall be used with, is selected in the baseElementClass field (see fig 9.6). We take the WorkItem Interface here, thus the strategy is used for all WorkItems (BugReports etc.).


New Extension

Extension Point Selection

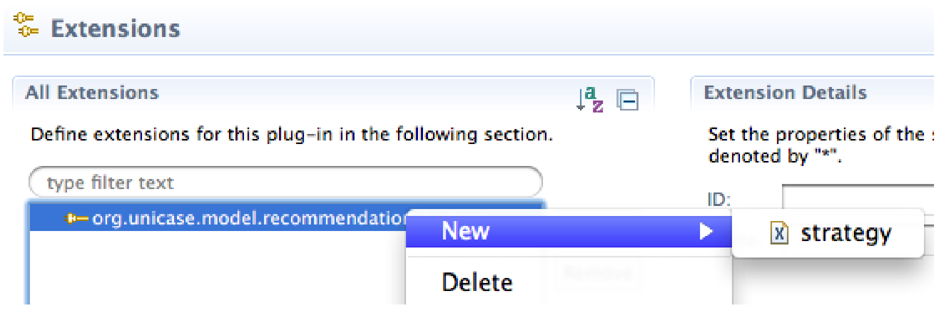
Create a new RecommendationStrategy extension.

Extension Point filter: org.unicase.model

org.unicase.model.recommendationstrategy

☑ Show only extension points from the required plug-ins

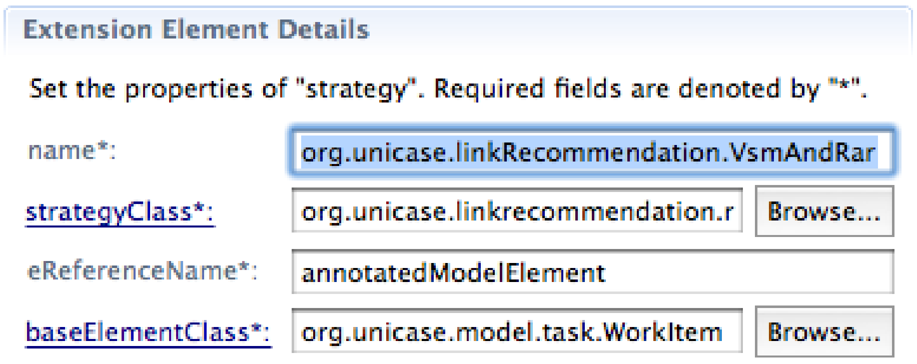
Figure 9.2.: Extension Manual: Part 2


Extensions

All Extensions

Define extensions for this plug-in in the following section.

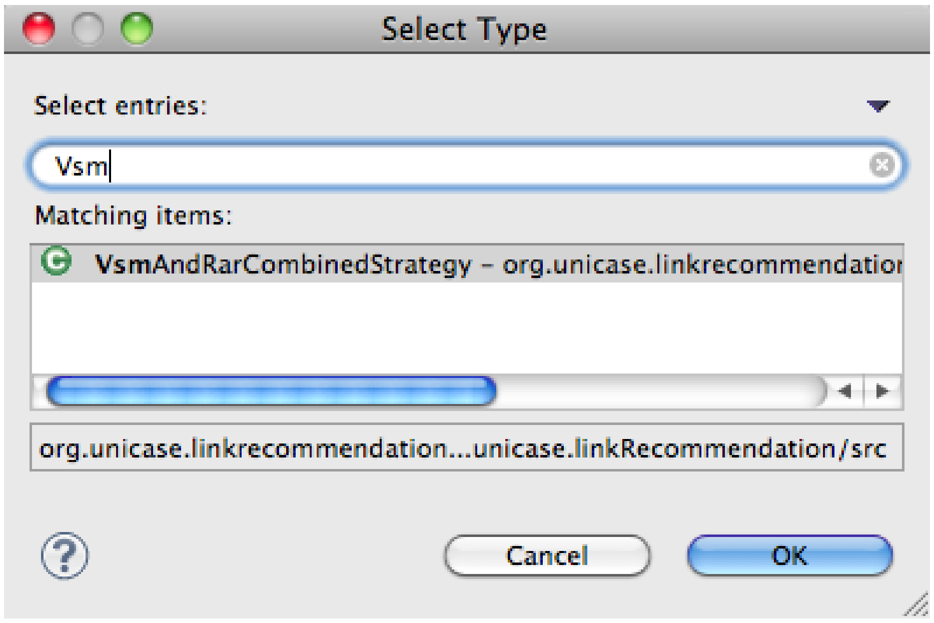
Figure 9.3.: Extension Manual: Part 3


Extension Element Details

Set the properties of "strategy". Required fields are denoted by "*".

name*:	org.unicase.linkRecommendation.VsmAndRar
strategyClass*:	org.unicase.linkrecommendation.r
eReferenceName*:	annotatedModelElement
baseElementClass*:	org.unicase.model.task.WorkItem

Figure 9.4.: Extension Manual: Part 4



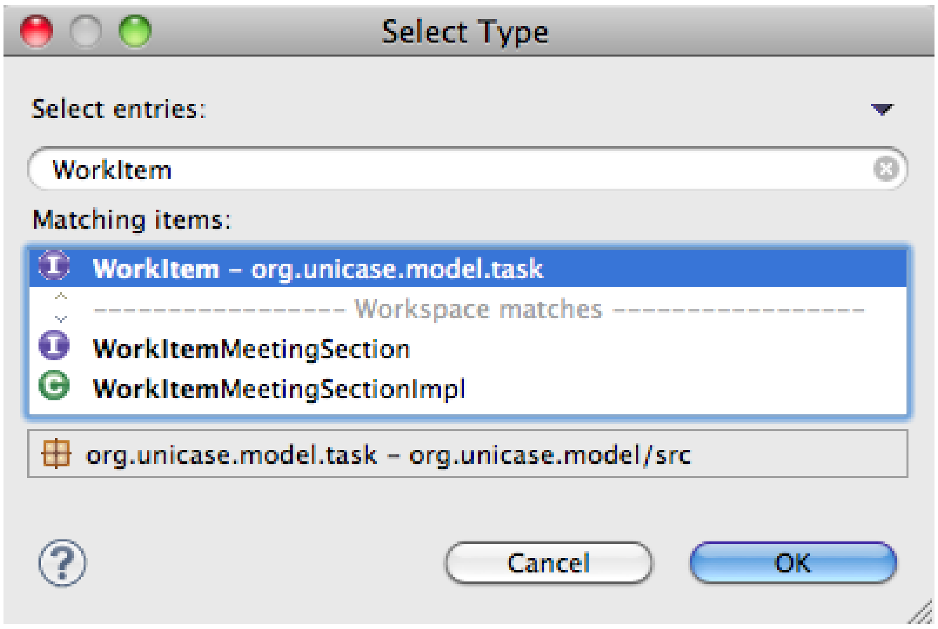
Vsm

Matching items:

VsmAndRarCombinedStrategy – org.unicase.linkrecommendatio

org.unicase.linkrecommendation...unicase.linkRecommendation/src

Figure 9.5.: Extension Manual: Part 5



WorkItem

Matching items:

WorkItem – org.unicase.model.task

---------------- Workspace matches ----------------

WorkItemMeetingSection

WorkItemMeetingSectionImpl

org.unicase.model.task – org.unicase.model/src

Figure 9.6.: Extension Manual: Part 6

Part IV.

Evaluation

10. Evaluation

≫ *First get your facts; then you can distort them at your leisure.* ≪

Mark Twain

This chapter evaluates the approaches used, analysing the different strategies listed in section 2.4. We used the two different projects UNICASE and DOLLI II for this evaluation. These are real projects with real users and real history data. We furthermore looked at two different kinds of links, to be explained in the next section. Afterwards we tried to find out, which strategies (recommendation and selection) are best for which use case. From there we derive our decisions of which strategies to use in the next deployment of UNICASE. The results presented underneath have been submitted to the ICSE2010 in [HFKB09].

10.1. Links under Consideration

For the question on which links we wanted to work for evaluation, we kept in mind two different aspects. Which links are useful and which links are represented in science, so that we can compare our results. Therefore we looked at two different kinds of links in detail.

10.1.1. Functional Requirement to Use Case

The first link models the relation between a functional requirement and a use case detailing the functional requirement. We have chosen this link, because it is common in several existing requirement models (e.g. [LOT08]). Functional requirements are text-based descriptions of certain functionalities of the system, while use cases describe an abstraction of a user interacting with the system. In UNICASE the relationship between functional requirements and use cases is $m : n$, meaning that a functional requirement can be detailed in more then one use case and a use case can detail several functional requirements. Furthermore requirements are structured in a hierarchy by the “refines” association (see figure 10.1).

10.1.2. Action Item to Functional Requirement

Action items are part of the project model and describe ongoing work in UNICASE. Action items can be linked to functional requirements to express that a certain action item needs to be processed to fulfil a certain functional requirement. This link can be traced by developers to find respective parts of the specification [HDKN09] and also to notify developers about relevant changes (see figure 10.2). We chose this link, because it is a good example of a link between project and system model.

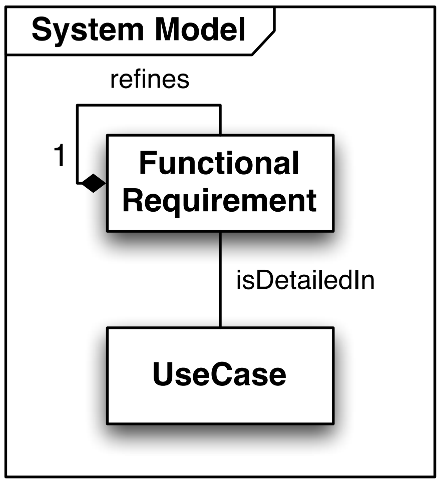


Figure 10.1.: Link between Functional Requirement and Use Case, UML class diagram

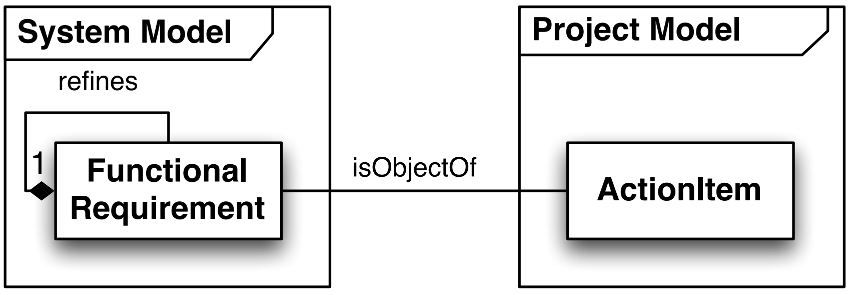


Figure 10.2.: Link between Action Items and Functional Requirements, UML class diagram

10.2. Analysed Projects

The data used for evaluation is taken from two real software projects, which have used UNICASE for modelling and management. One of which is the development project of UNICASE itself, with a history of nearly 3000 versions and 26 participants over nearly a year. The second one is DOLLI II, a student project with 25 participating students over 6 months and approximately 650 versions; moreover, DOLLI II's main language is German. We did not apply stemming in the DOLLI project. In both projects, users explicitly maintained traceability links. We can use these existing traceability links for our evaluation by comparing them with the recommendations from different approaches.

As all the links of interest are bidirectional, we were able to apply the recommendation from both directions, doubling the candidates for the evaluation. In total can evaluate the following number of links:

#Links	UNICASE-Project	DOLLI II-Project
AI ↔ FR	288 (144 each)	362 (181 each)
FR ↔ UC	108 (54 each)	50 (25 each)

Table 10.1.: Evaluation: Number of Links in Total

10.3. Methods for Evaluation

In order to evaluate the proposed strategies adequately we created two different cases of evaluation. To assess the link recovery use case we analysed the most recent state of the system (state-based evaluation). For the use case link recommendation, we iterate over the project history and apply the recommendation on exactly the project state, in which the user has created the original link (history-based evaluation). We compare these suggestions with the link existing or being created at this state of the project.

10.3.1. State-based Evaluation

For the state-based evaluation the evaluation projects have been assessed in the latest version. All existing links of the relevant type are considered. The recommendation strategies were applied on both linked artifacts of these link instances. Afterwards the recommendation was compared to the existing links and the metrics, according to section 2.6.2, were calculated. We additionally recorded the average position in the recommendation. In order to retrieve valid results, the existing link was ignored in recommendation processes for all structure-based approaches.

10.3.2. History-based Evaluation

The state-based evaluation we presented before is suitable to evaluate different approaches for the link recovery use case. For the use case of link recommendation, the results would be biased, because the recommendation is based on another project state compared to the state, when the user actually wanted to create a specific link. This is especially true for approaches like ARM, which are based on the project history. Therefore we suggest a novel approach of evaluation, which we call history-based evaluation. In this approach, we iterate through the project history until we find a moment, where the user created a link. At every link creation operation, we make a recommendation based on the state of the project at this very moment. Then again we compare the result of this recommendation with the choice of the user. We claim this way of evaluation to be more realistic then the state-based evaluation. We reapplied the most successful strategies, VSM and VSM+RAR of the state-based evaluation as well as the ARM approach on the UNICASE project for both link types.

10.4. State-based Evaluation

In a first step, we compare the two content-based approaches LSI and VSM. Based on the research shown in this section, we found VSM the better working strategy for further evaluation. In a second step, we compare different settings for the selection strategies from section 2.4. Using the best set-up for the selection strategies, we evaluate ARM as well as the two structure-based approaches SRR and RAR. As a last step of the state-based evaluation we compare combinations of the most successful strategies. In the second part, the history-based evaluation, we apply VSM and the two most successful combinations of strategies for the recommendation use case.

10.4.1. General Findings

We applied VSM from section 2.4 on the UNICASE case-study to get an overview of the limits in our system due to lack of or improper documentation within the data.

Threshold = 0.5	UNICASE		DOLLI II	
	Precision	Recall	Precision	Recall
AI ↔ FR	0.53	0.25	0.54	0.04
FR ↔ UC	0.6	0.27	0.5	0.02

Table 10.2.: Evaluation: VSM in General

Comparing these results with related work results of other projects, values might seem relatively low. This is due to the fact that the proposed links are fine-grained traceability links on the level of model elements rather then coarse-grained links among relatively large documents. Therefore elements under consideration often have very few textual content. Consequently content-based approaches provide lower results in this context. Secondly our values are calculated based on the selection made by the developers in the projects. It contains potential flaws which might result in low values: On one hand the system might make correct suggestions, where the (correct and suggested) link is not established in the project. On the other hand incorrect links in the project might distort the final values and lead to lower values than expected. Because of the different granularity of link recommendation, our results can not be directly compared to the results of other authors. To close this gap, we therefore compare different extensions and strategies in our context.

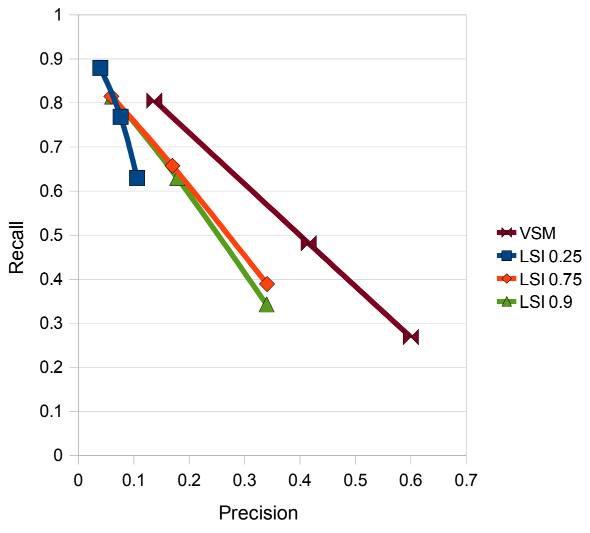


Figure 10.3.: FR ↔ UC, UNICASE

10.4.2. Latent Semantic Indexing (LSI)

To analyse the application of LSI in the unified model, we decided on the functional requirements to use case link in the UNICASE project and compared it with VSM. The importance, but also difficulty, in determining the factor k in LSI is still an open issue in science (see e.g. [LD06]). Hence we tried various values for k and compared the results (see figure 10.3).

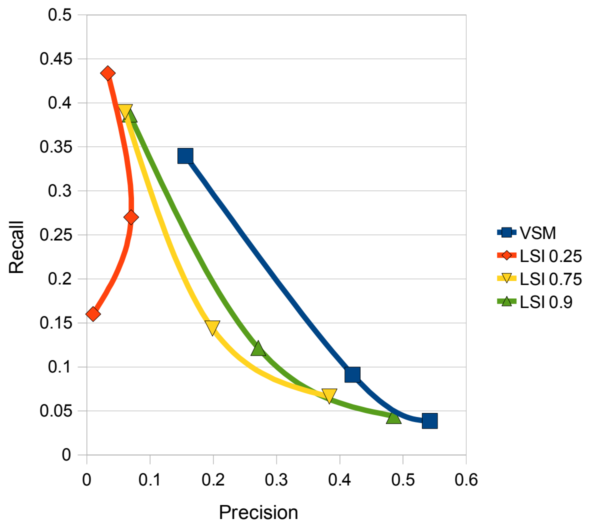


Figure 10.4.: AI ↔ FR, DOLLI II

The results in the DOLLI II project (see figure 10.4) were worse than in the UNICASE project, as we did not apply any stemming. In both of the projects we could not confirm the strong advantages of LSI, which have been proposed by other authors like [ACCDL00] or [LD06]. Similar to the case studies presented in [HDS06] or [Lor05] this relatively bad performance of LSI might also result from the small number of text and words appearing. As a conclusion, together with the calculation complexity of LSI, we decided to stick to VSM in our further analysis.

10.4.3. The Selection

We chose both the VSM approach and one of the other strategies from our science to compare cut point with threshold selection in the UNICASE project on the requirement to use case link:

Threshold	Avg. Position	Precision	Recall
T=0.1	3.94	0.14	0.81
T=0.35	1.9	0.42	0.48
T=0.5	1.3	0.6	0.27

Table 10.3.: Evaluation: VSM, Threshold, FR ↔ UC, UNICASE

Cut Point	Avg. Position	Precision	Recall
CP=5	3.09	0.2	0.69
CP=10	4.35	0.12	0.76

Table 10.4.: Evaluation: VSM, Cut Point, FR ↔ UC, UNICASE

RAR	T=0.1		T=0.35		T=0.5	
	Prec.	Rec.	Prec.	Rec.	Prec.	Rec.
UNICASE	0.05	0.43	0.12	0.21	0.16	0.19
DOLLI II	0.03	0.36	0.04	0.31	0.07	0.23

Table 10.5.: Evaluation: RAR, Threshold, FR ↔ UC

RAR	Cut Point=5		Cut Point=10	
	Prec.	Rec.	Prec.	Rec.
UNICASE	0.08	0.35	0.05	0.43
DOLLI II	0.07	0.25	0.05	0.29

Table 10.6.: Evaluation: RAR, Cut Point, FR ↔ UC

Analysing these results, we do not see a significant impact of the selection strategies in the VSM approach; whereas in the RAR approach it is significantly easier to select a proper value using cut point strategy. This behaviour comes from the following: The weighting function in RAR is relative to the results. RAR has too little information to set absolute probabilities. In contrast, VSM is able to set absolute weights, hence e.g. the value of 0.5 in VSM is an absolute assertion, comparable in every case. In contrast, a value of 0.5 in RAR just says, that elements with a higher value are better, but the value itself is just a tendency, not an absolute probability.

10.4.4. History-Based Approach

Association Rule Mining (ARM)

We then started to evaluate the different approaches and combinations. We first of all tried to extract the information content from the project history with the Association Rule Mining (ARM) strategy.

	Avg. Position	Precision	Recall
AI → FR	1.32	0.33	0.28
FR → UC	1.65	0.65	0.2

Table 10.7.: Evaluation: ARM, CP=10, UNICASE

However, these results need to be handled with care, as the ARM strategy was trained with the whole project history. Linked artifacts were likely to be changed in the same session because users could have navigated that link. For this reason we used the history-based evaluation strategy and only considered the history before the link was created.

10.4.5. Structural Approaches

In addition to the widely accepted content- and text-based approaches like VSM and LSI and the ARM approach from our studies, we reckoned that other strategies might include information which can be extracted; moreover, that these structure-based strategies might provide benefit to a combined solution. Analysing these approaches is a non trivial task. To guarantee scientific results, it must be ensured that the correct (and usually already established link) does not contribute to the recommendation. To solve this problem and hence produce reasonable results our algorithms include the possibility to ignore certain links. Another impediment in evaluation is the semantics behind these approaches. Developers, who already worked on tasks to a certain functional requirement are more probable to be asserted to other action items related to these; more abstract: semantics is used to gain knowledge or assumptions are made that indicate that the elements suggested are not equally probable. Consequently these strategies can only be used in this specific semantic context and thus lead to a smaller set of tests. In our concrete case this means that we could apply the Related Assignees Recommendation (RAR) only in one way.

Shared References Recommendation (SRR)

SRR considers the artifacts, which are already linked to an element. We wanted to evaluate how deep references should be searched through and how the weighting should be set. We decided for the above-mentioned weighting, normalised at the maximum and took the cut point selection to reduce the importance of this measurement. First we analysed how values changed in the UNICASE project over iteration depth:

Depth	Avg. Position	Precision	Recall
Depth=1	3.85	0.02	0.07
Depth=2	9.83	0.01	0.13
Depth=3	8.69	0.03	0.33
Depth=4	10.01	0.01	0.09
Depth=5	9.88	0.01	0.13

Table 10.8.: Evaluation: SRR, CP=10, AI $\rightarrow$ FR, UNICASE

The studies suggested that three was the clear local optimum. A detailed investigation showed, that the strategy does not work well for elements, which are not linked at all and those which are too widely spread. We realized that elements depending on their linking had different sizes of clouds. Hence we changed the algorithm, so that the clouds do not go for a certain radius (which is the depth), but keep iterating until a certain size is reached (measured by the number of elements found) or the cloud does not grow anymore, i.e. the set is the transitive closure. We analysed four different stages, depending on the size of the semantic clouds:

Size	Avg. Position	Precision	Recall
Size=10	9.86	0.01	0.13
Size=25	8.47	0.03	0.32
Size=50	8.46	0.04	0.35
Size=75	8.63	0.03	0.35

Table 10.9.: Evaluation: SRR with Semantic Clouds, CP=10, AI $\rightarrow$ FR, UNICASE

Analogue to the cut point strategy, we have thus a possibility at hand to limit the number of suggestions. We see that the average position is significantly lower than in the suggestions before. Also the recall stays approximately the same.

Related Assignees Recommendation (RAR)

The second structural approach we developed was RAR, which we applied on both projects (see 10.6).

These fairly low values result from the fact that there might always be many related items, and also often cases in which there is no information content available through this method. It also very much depends on the assignment behaviour in this particular project. Nevertheless these statistics show, that there can be information content retrieved through this approach as the values listed above are way over guessing average on 181 or 144 examples. However this solution is not a satisfying recommendation on its own.

10.4.6. Combining Strategies

Based on our results for both content-based and structural algorithms, we evaluated different combinations. Based on the assumption that each high value in one of underlying recommendations should lead to suggestion we composed two strategies into one by taking the maximum of both values.

According to the results presented above we combined VSM with RAR and SRR with cloud size of 50 elements.

Strategy	Precision	Recall
VSM	0.61	0.17
Max: VSM+SRR(50)	0.02	0.67
Max: VSM+RAR	0.24	0.35

Table 10.10.: Evaluation: T=0.5, AI → FR, UNICASE

	UNICASE		DOLLI II	
	Prec.	Rec.	Prec.	Rec.
VSM	0.07	0.65	0.07	0.38
Max: VSM+SRR(50)	0.04	0.37	0.05	0.41
Max: VSM+RAR	0.08	0.76	0.06	0.45

Table 10.11.: Evaluation: CP=10, AI → FR

The statistics suggest that taking the maximum can lead to entirely different results. This is due to the weighting function of the structure-based approaches. They are rather relative values, which can not be used in an equal manner as the absolute values returned by the VSM-Strategy. Nevertheless these statistics also show that the problem can be overcome by combining the maximum function with a cut point strategy. However, these results reveal that the precision of the structural approaches was too low to justify taking the maximum; still, used with two strategies returning good absolute results, this strategy can be an option.

In a second step we combine two strategies with a certain factor, of which the most obvious would be 1:1, leading to the arithmetic mean. Based on our preliminary results, we used the cut point strategy:

Strategy	Precision	Recall
VSM	0.61	0.17
Max: VSM+SRR(50)	0.09	0.39
Max: VSM+RAR	0.37	0.22

Table 10.12.: Evaluation: Avg.Mean, T=0.5, AI → FR, UNICASE

	UNICASE		DOLLI II	
	Prec.	Rec.	Prec.	Rec.
VSM	0.07	0.65	0.07	0.38
Avg: VSM+SRR(50)	0.05	0.53	0.05	0.49
Avg: VSM+RAR	0.08	0.76	0.06	0.49

Table 10.13.: Evaluation: Avg.Mean, CP=10, AI → FR

They also indicate that the combination of VSM and RAR used with cut point selection can improve the results. We wanted to evaluate, whether the results would increase by giving the VSM a higher importance. Therefore we tried VSM+RAR on UNICASE with different factors and had a look at the results:

	CP=5			CP=10	
	Avg Pos.	Precision	Recall	Precision	Recall
VSM	3.74	0.12	0.54	0.07	0.65
0.1 – 0.9	3.91	0.11	0.56	0.08	0.74
0.35 – 0.65	3.66	0.12	0.6	0.08	0.78
0.5 – 0.5	3.49	0.13	0.66	0.08	0.76
0.75 – 0.25	3.31	0.13	0.65	0.08	0.75

Table 10.14.: Evaluation: VSM+RAR, CP=10, AI → FR, UNICASE

To sum up this section, we can see that we can improve the recall and also precision by combining different strategies. A remaining problem is the selection of a function for the combination of different strategies.

10.4.7. Optimisations

All results shown above include optimizations included during the development. They were analysed separately.

CamelCase Extraction

Many times, in the project model as well as in the system model, java-like upper- or lower-camel case notation is used in descriptions and titles. To get the information content from these words, they need to be split. Whereas in most cases this is a trivial update, it has to be used with care, e.g. in uppercase words like abbreviations. This rather easy implementable optimisation brings quite a big benefit to the overall result:

	Avg.Pos.	Precision	Recall
No Camel Case Extraction	5.72	0.07	0.59
Camel Case Extraction	5.29	0.07	0.65

Table 10.15.: Evaluation: CamelCase-Extraction

Empowering the name of an object

Another improvement addressed the difference between text from the name and the description of a model element. To make sure, that the title is more important than the description, it was tripled in all calculations.

We evaluated this improvement with both earlier mentioned links:

	Precision	Recall
regular quantification	0.47	0.28
tripling the name	0.52	0.46

Table 10.16.: Evaluation: Quantification, VSM, T=0.35, FR $\leftrightarrow$ UC

	AI $\leftrightarrow$ FR	FR $\leftrightarrow$ UC
regular quantification	5.32	4.54
tripling the name	4.63	4.61

Table 10.17.: Evaluation: Quantification, VSM, CP=10, Avg. Position

Statistics show that this improvement can give a benefit to the overall result; however, in some cases the correct elements are ranked a little lower in suggestions. This is probably due to elements with poor textual descriptions in the title.

10.5. History-based Evaluation

The state-based evaluation we presented before is suitable to evaluate different approaches for the link recovery use case. For the use case of link recommendation the results would be biased, as the recommendation is based on another project state compared to the state when the user actually wanted to create a specific link. This is especially true for approaches like ARM, which are based on the project history. Therefore we suggest a novel approach of evaluation, called history-based evaluation. In this approach, we iterate through the project history until the user creates a link. At every link creation operation, we make a recommendation based on the state the project was at. Then again we compare the result of this recommendation with the choice of the user. We claim this way of evaluation to be more realistic then the state-based evaluation. We reapplied the most successful strategies, VSM and VSM+RAR of the state-based evaluation as well as the ARM approach on the UNICASE project for both link types. We applied this approach for the UNICASE project:

	Method	VSM		Avg.:VSM + SRR(Size: 50)	
		Precision	Recall	Precision	Recall
AI ↔ FR	history	0.12	0.54	0.07	0.65
	state	0.11	0.56	0.08	0.74
FR ↔ UC	history	0.12	0.6	0.08	0.78
	state	0.13	0.66	0.08	0.76

Table 10.18.: Evaluation: Evaluation Methods, CP=10, UNICASE

Evaluation Method	Precision	Recall
history-based	0.07	0.61
state-based	0.08	0.76

Table 10.19.: Evaluation: Avg.Mean: VSM+RAR, CP=10, AI → FR, UNICASE

We observe differences between the values of the history- and the state-based evaluation. We claim that a history-based evaluation is a more difficult but also realistic scenario for the use case of link recommendation. This is especially true for structural and history-based approaches. We can observe that especially the structure-based approaches lead to lower results, this is due to the incomplete structure at the project state of the recommendation. We can further observe that links between use cases and functional requirements are recommended much worse, also compared to the second link between functional requirements and action items. An explanation for this observation could be that functional requirements and action items are usually filled with content, when they are linked, while especially use cases are often filled with content after the link to a functional requirement is created.

As discussed in section 10.4.4 we wanted to deepen the evaluation of the ARM approach over project history. We realized that the values over time were quite low. Just three elements out of 88 could be recognised, as just for a handful of link suggestions were made. At second view we realised, that most links were created in the same session as one or both of the linked artifacts. As in the history-based approach only the user behaviour from the time before the linking event is mined by ARM, the results were very low. This reveals a drawback of the ARM strategy; it only works for elements which are existent in the system for a certain time. Therefore the ARM approach has its strength in traceability link recovery, but not in recommendation, based on our observed user behaviour to create links quite early.

10.6. Conclusion

Revising the evaluation we can see three major results: First of all, our optimisations were justified. We saw that the little improvements we made, supported the recommendation in terms of precision and recall. Thus all optimisations were kept in the framework. Secondly, one could see that the novel approach of evaluation, the history-based evaluation, does make a big difference in analysing the quality of link recommendation. Lastly, we do now know that the combined approach of the vector space model and the related assignees recommendation is the most successful of the strategies. However, as this approach can only be applied on a single link type, we choose the VSM approach for other links. We chose this for reasons of performance, as the vector space model can compute results in short time and as the evaluation proved, with a good precision/recall proportion. According to the selection part of this evaluation, we furthermore decide to use threshold selection with a high threshold for the link recovery use case, as this was most promising in precision. Remarkable is also that on our data, latent semantic indexing does not show the remarkable advantages proposed by others (e.g. [ACCDL00]).

11. Outlook and Conclusion

≫ *Success is like a pie, there are different layers* ≪

Qun Zhang, Chinese Poet

This chapter summarises this thesis and indicates in which parts further research can extend the presented work.

11.1. Conclusion

This thesis presented the idea of traceability and suggested to use traceability link recommendation and recovery. It showed that to a certain extend, the latter can improve the current use of traceability, although a perfect solution seems to be unachievable.

This thesis explained the strategies in use for traceability recovery like Vector Space Model (VSM) and Latent Semantic Indexing (LSI). It also introduced new strategies based on recent science like Association Rule Mining (ARM) and explained completely new approaches based on the structure of the project, namely Related Assignees Recommendation (RAR) and Shared References Recommendation (SRR).

Requirements to a system for link recommendation and recovery were proposed and a prototype framework was developed. It provides functionality for third-party-developers to extend and alternate the strategies in use. The framework also offers the possibility for third-party-developers to introduce very specialised algorithms for particular links and model elements. Additionally the framework provides an opportunity to use combinations of recommendation strategies. Also implementations for the approaches mentioned above were created.

Using this framework, important questions were analysed and answered through evaluation with two major software projects from recent studies. Besides the evaluation of the introduced algorithms, we evaluated questions from recent work. Additionally we introduced a novel approach for evaluation, which simulates a more realistic link recommendation scenario. We found that reasonable support for creating traceability links can be provided, although a perfect solution is out of reach. This is due to single badly documented artifacts as well as the problem, that less documentation is needed (and hence provided) in a model based approach like UNICASE. However, we found that VSM can be improved by one of our structural approaches and that in our case latent semantic indexing does not have the advantages proposed by authors from related work. The ARM approach was applied unsuccessfully, as it came out that the information ARM needs to make appropriate suggestions, is mostly created after the suggestion is demanded.

The prototype implements features for full traceability recommendation and recovery support. It was included into UNICASE and is now part of the deployed version of UNICASE.

11.2. Outlook

Further improvements have to and will be made in the development of strategies for link recommendation and recovery; hence, we provided the framework with extensibility for these strategies. Especially the structure- and history-based approaches need further research. In our opinion the essential but very difficult point is the weighting function for structural and history based approaches. Also more sophisticated stemming algorithms can improve the results.

Refinements can furthermore be created by specialising the strategies to the single link and artifact types. The RAR approach was one step in that direction. As mentioned in the previous chapter, different weighting functions for structure based approaches could provide enhancements as well.

Another area for development is learning from user feedback. Related work showed that results can be improved by including user behaviour into the calculations. One option for this can be the construction of artificial neural networks trained with recent behaviour.

Part V.

Appendix

List of Figures

List of Tables

Bibliography

[Abb83] R. J Abbott. Program design by informal english descriptions. *ACM New York, NY, USA*, 1983.

[ACC$^+$00a] G. Antoniol, G. Canfora, G. Casazza, A. De Lucia, and E. Merlo. Tracing object-oriented code into functional requirements. In *Program Comprehension, 2000. Proceedings. IWPC 2000. 8th International Workshop on*, pages 79–86, 2000.

[ACC00b] G. Antoniol, C. Casazza, and A. Cimitile. Traceability recovery by modeling programmer behavior. In *Proceedings Seventh Working Conference on Reverse Engineering*, pages 240–247, Brisbane, Qld., Australia, 2000.

[ACC$^+$02] G. Antoniol, G. Canfora, G. Casazza, A. De Lucia, and E. Merlo. Recovering traceability links between code and documentation. *Software Engineering, IEEE Transactions on*, 28(10):970–983, 2002.

[ACCDL00] G. Antoniol, G. Canfora, G. Casazza, and A. De Lucia. Identifying the starting impact set of a maintenance request: a case study. In *Software Maintenance and Reengineering, 2000. Proceedings of the Fourth European*, pages 227–230, 2000.

[ACCL00] G. Antoniol, G. Canfora, G. Casazza, and A. De Lucia. Information retrieval models for recovering traceability linksbetween code and documentation. In *Software Maintenance, 2000. Proceedings. International Conference on*, pages 40–49, 2000.

[ACLM99] G. Antoniol, G. Canfora, A. De Lucia, and E. Merlo. Recovering code to documentation links in oo systems. In *Proc. of the Working Conference on Reverse Engineering*, pages 136–144, 1999.

[Aut09a] Various Authors. unicase - detailed description. `http://teambruegge.informatik.tu-muenchen.de/groups/unicase/wiki/7c90d/unicase_-_Detailed_description.html`, September 2009.

[Aut09b] Various Authors. What is unicase? `http://teambruegge.informatik.tu-muenchen.de/groups/unicase/wiki/bdc81/What_is_unicase_.html`, September 2009.

[BCHK08] B. Bruegge, O. Creighton, J. Helming, and M. Kogel. Unicase-an ecosystem for unified software engineering research tools. In *Third IEEE International Conference on Global Software Engineering, ICGSE*, 2008.

[BD03] B. Bruegge and A.H. Dutoit. *Object-Oriented Software Engineering: Using UML, Patterns and Java.* Prentice-Hall, Inc. Upper Saddle River, NJ, USA, 2003.

[Bri03] LC Briand. Impact Analysis and Change Management of UML Models. In *icsm*, page 256. IEEE Computer Society, 2003.

[CJ90] E. J. Chikofsky and II JH. Reverse engineering and design recovery: A taxonomy. *IEEE software*, 7(1):13–17, 1990.

[CSDZ05] J. Cleland-Huang, R. Settimi, Chuan Duan, and Xuchang Zou. Utilizing supporting evidence to improve dynamic requirements traceability. In *13th IEEE International Conference on Requirements Engineering (RE'05)*, pages 135–144, Paris, France, 2005.

[Dav08] J. David. Recommending software artifacts from repository transactions. *Lecture Notes in Computer Science*, 5027:189–198, 2008.

[DDF$^+$90] S. Deerwester, S. T Dumais, G. W Furnas, T. K Landauer, and R. Harshman. Indexing by latent semantic analysis. *Journal of the American society for information science*, 41(6):391–407, 1990.

[DKNH09] J. David, M. Koegel, H. Naughton, and J. Helming. Traceability ReARMed. In *2009 33rd Annual IEEE International Computer Software and Applications Conference*, 2009.

[DP98] R. Doemges and K. Pohl. Adapting traceability environments to project-specific needs. *ACM New York, NY, USA*, 1998.

[GB03] E. Gamma and K. Beck. *Contributing to Eclipse: Principles, Patterns, and Plugins.* Addison Wesley Longman Publishing Co., Inc. Redwood City, CA, USA, 2003.

[GF94] O. Gotel and A. Finkelstein. An analysis of the requirements traceability problem. In *Proceedings of the First International Conference on Requirements Engineering*, pages 94–101, 1994.

[GHJ98] H. Gall, K. Hajek, and M. Jazayeri. Detection of logical coupling based on product release history. In *Proceedings of the 14th IEEE International Conference in Software Maintainance*, 1998.

[GJK03] H. Gall, M. Jazayeri, and J. Krajewski. Cvs release history data for detecting logical couplings. In *Software Evolution, 2003. Proceedings. Sixth International Workshop on Principles of*, pages 13–23, 2003.

[Gra92] R. B Grady. *Practical software metrics for project management and process improvement.* Prentice-Hall, Inc. Upper Saddle River, NJ, USA, 1992.

[HDKN09] J. Helming, J. David, M. Koegel, and H. Naughton. Integrating system modeling with project management–a case study. In *International Computer Software and Applications Conference, COMPSAC 2009*, 2009.

[HDO03] J.H. Hayes, A. Dekhtyar, and J. Osborne. Improving requirements tracing via information retrieval. In *Requirements Engineering Conference, 2003. Proceedings. 11th IEEE International*, pages 138–147, 2003.

[HDS06] J.H. Hayes, A. Dekhtyar, and S.K. Sundaram. Advancing candidate link generation for requirements tracing: the study of methods. *IEEE Transactions on Software Engineering*, 32(1):4–19, 2006.

[HDSH04] J. H. Hayes, A. Dekhtyar, S. K. Sundaram, and S. Howard. Helping analysts trace requirements: An objective look. In *12th IEEE International Requirements Engineering Conference, 2004. Proceedings*, pages 249–259, 2004.

[HFKB09] J. Helming, H. Femmer, M. Koegel, and B. Bruegge. Recommendation and recovery of traceability links. Submitted to ICSE2010, 2009.

[KGFK03] A. Von Knethen, M. Grund, I. Fraunhofer, and G. Kaiserslautern. QuaTrace: a tool environment for (semi-) automatic impact analysis based on traces. In *Software Maintenance, 2003. ICSM 2003. Proceedings. International Conference on*, page 246–255, 2003.

[LD06] M. Lormans and A. Van Deursen. Can LSI help reconstructing requirements traceability in design and test? In *Software Maintenance and Reengineering, 2006. CSMR 2006. Proceedings of the 10th European Conference on*, 2006.

[LFOT04] A. De Lucia, F. Fasano, R. Oliveto, and G. Tortora. Enhancing an artefact management system with traceability recovery features. In *20th IEEE International Conference on Software Maintenance, 2004. Proceedings*, pages 306–315, 2004.

[LFOT07] A. De Lucia, F. Fasano, R. Oliveto, and G. Tortora. Recovering traceability links in software artifact management systems using information retrieval methods. *ACM Transactions on Software Engineering and Methodology*, 16(4), 2007.

[Lor05] M. Lormans. Reconstructing requirements coverage views from design and test using traceability recovery via LSI. In *Proceedings of the 3rd international workshop on Traceability in emerging forms of software engineering*, pages 37–42, 2005.

[LOT08] A. De Lucia, R. Oliveto, and G. Tortora. Adams re-trace: traceability link recovery via latent semantic indexing. In *Proceedings of the 30th international conference on Software engineering*, pages 839–842. ACM New York, NY, USA, 2008.

[Mal09] Elwyn Malethan. Humanise CamelCase in java. `http://www.malethan.com/article/humanise_camel_case_in_java.html`, October 2009.

[MM03] A. Marcus and J.I. Maletic. Recovering documentation-to-source-code traceability links using latent semantic indexing. In *Software Engineering, 2003. Proceedings. 25th International Conference on*, pages 125–135, 2003.

[MNS01] G.C. Murphy, D. Notkin, and K.J. Sullivan. Software reflexion models: bridging the gap between design and implementation. *IEEE Transactions on Software Engineering*, 27(4):364–380, 2001.

[oDRC$^+$02] J. Natt och Dag, B. Regnell, P. Carlshamre, M. Andersson, and J. Karlsson. A feasibility study of automated natural language requirements analysis in market-driven development. *Requirements Engineering*, 7(1):20–33, 2002.

[RGP86] C. V. Ramamoorthy, V. Garg, and A. Prakash. Programming in the large. *IEEE Transactions on Software Engineering*, 72(7):769–783, 1986.

[Rou06] W. Roush. Marvin minsky on common sense and computers that emote. *Technology Review, July*, 2006.

[SCK$^+$04] R. Settimi, J. Cleland-Huang, O. Ben Khadra, J. Mody, W. Lukasik, and C. DePalma. Supporting software evolution through dynamically retrieving traces to UML artifacts. In *Proceedings. 7th International Workshop on Principles of Software Evolution, 2004.*, pages 49–54, Kyoto, Japan, 2004.

[Sin01] A. Singhal. Modern information retrieval: A brief overview. *IEEE Data Engineering Bulletin*, 24(4):35–43, 2001.

[SSC96] M. Sefika, A. Sane, and R.H. Campbell. Monitoring compliance of a software system with its high-level design models. In *Proceedings ICSE*, volume 18, pages 387–396, 1996.

[Vog09] Lars Vogel. Eclipse extension points and extensions - tutorial. http://www.vogella.de/articles/EclipseExtensionPoint/article.html, September 2009.

[Wri91] S. Wright. Requirements Traceability-What? why? and how?, tools and techniques for maintaining traceability during design. In *IEEE Colloquium, Computing and Control Division, Professional Group C*, volume 1, pages 1–1, 1991.

[YMN$^+$04] A. T. T. Ying, G. C. Murphy, R. Ng, M. C. Chu-Carroll, I. B.M.T.J.W.R Center, and N. Y. Hawthorne. Predicting source code changes by mining change history. *IEEE Transactions on Software Engineering*, 30(9):574–586, 2004.

[ZWDZ05] T. Zimmermann, P. Weissgerber, S. Diehl, and A. Zeller. Mining version histories to guide software changes. *IEEE Transactions on Software Engineering*, pages 429–445, 2005.