Software Case Similarity Measure

ReDSeeDS project

Deliverable D4.2, version 1.0, 31.07.2007

IST-2006-033596
ReDSeeDS
Requirements Driven
Software Development System
www.redseeds.eu

Infovide S.A., Poland
Warsaw University of Technology, Poland
Hamburger Informatik Technologie Center e.V., Germany
University of Koblenz-Landau, Germany
University of Latvia, Latvia
Vienna University of Technology, Austria
Fraunhofer IESE, Germany
Algoritmu sistemos, UAB, Lithuania
Cybersoft IT Ltd., Turkey
PRO DV Software AG, Germany
Heriot-Watt University, United Kingdom

Software Case Similarity Measure
ReDSeeDS project

Workpackage	WP4
Task	T4.2
Document number	D4.2
Document type	Deliverable
Title	Software Case Similarity Measure
Subtitle	ReDSeeDS project
Author(s)	Katharina Wolter, Thorsten Krebs, Daniel Bildhauer, Markus Nick, Lothar Hotz
Internal Reviewer(s)	Albert Ambroziewicz, Jacek Bojarski, Wiktor Nowakowski, Tomasz Straszak, Kizito Ssamula Mukasa, Robert Pooley
Internal Acceptance	Project Board
Location	https://svn.redseeds.eu/svn/redseeds/1_DeliverablesSpace/WP4_-Technologies_for_reusable_cases/D4.2.00/ReDSeeDS_D4.2_Software_Case_Similarity_Measure_Definition.tex
Version	1.0
Status	Final
Distribution	Public

31.07.2007

History of changes

Date	Ver.	Author(s)	Change description
13.06.2007	0.01	Katharina Wolter (UH)	Document creation and proposition of TOC
18.06.2007	0.02	Markus Nick (Fraunhofer)	First content for Sections 3.2 and 4.1
19.06.2007	0.03	Katharina Wolter (UH)	Similarity measure requirements
21.06.2007	0.04	Katharina Wolter (UH)	Further similarity measure requirements and figures
27.06.2007	0.05	Katharina Wolter (UH)	First content and figures for Section 4.1, further similarity measure requirements
29.06.2007	0.06	Katharina Wolter (UH)	Added content and figures for Section 4.1
01.07.2007	0.07	Thorsten Krebs (UH)	First contents for Section 3.3
03.07.2007	0.08	Katharina Wolter (UH)	Content and figure for Chapter 2
04.07.2007	0.08	Daniel Bildhauer (UKo)	First content for Section 3.4
04.07.2007	0.09	Katharina Wolter (UH)	Introduction for Chapter 3
06.07.2007	0.10	Markus Nick (Fraunhofer)	Section 3.1, Update of Section 3.2
09.07.2007	0.11	Katharina Wolter (UH)	First content for Section 4.3
10.07.2007	0.12	Katharina Wolter (UH)	Restructuring: split Chapter 4 in Chapter 4-6
11.07.2007	0.13	Thorsten Krebs (UH)	Reading & commenting plus more input for Section 3.3
11.07.2007	0.14	Daniel Bildhauer (UKo)	Initial content for section 4.4 and 4.6
11.07.2007	0.15	Katharina Wolter (UH)	Improved Chapter 4 and 5
13.07.2007	0.16	Katharina Wolter (UH)	Content for Section 5.4
17.07.2007	0.17	Thorsten Krebs (UH)	Filled Section 6.3
17.07.2007	0.18	Markus Nick (IESE)	Global similarity measure
17.07.2007	0.19	Katharina Wolter (UH)	Added content for Section 5.4

Date	Ver.	Author(s)	Change description
18.07.2007	0.20	Thorsten Krebs (UH)	Additional input (incl. figures) for the example in Section 6.3
18.07.2007	0.21	Katharina Wolter (UH)	Additional content for Sections 5.4, 3.1
19.07.2007	0.22	Katharina Wolter (UH)	Initial content for Section 6.1, small corrections and references in Chapter 3
19.07.2007	0.23	Daniel Bildhauer (UKo)	Initial content for sections 6.5 and 7.1.4
19.07.2007	0.24	Markus Nick (IESE)	Textual similarity measures, some general information on IR and CBR in Chapter 3
20.07.2007	0.25	Daniel Bildhauer (UKo)	More content for section 7.1.4, some cleanups
20.07.2007	0.26	Katharina Wolter (UH)	First content for Sec. 6.2 and Chap. 1, add. content for Sec. 3.1 and Chap. 4
23.07.2007	0.27	Daniel Bildhauer (UKo)	Refactoring of section 7.1.4 and 6.5
24.07.2007	0.28	Katharina Wolter (UH)	Content for Conclusion, add. content for Chap. 1
25.07.2007	0.29	Katharina Wolter (UH)	Add. content for Chap. 1 and Sec. 6.2, restructuring of Chap. 7, small corrections in Chap.4
25.07.2007	0.30	Lothar Hotz (UH)	Added content for Chapter 7
26.07.2007	0.31	Katharina Wolter (UH)	Restructuring of Sec. 5.5 and small corrections
26.07.2007	0.32	Katharina Wolter (UH)	Restructuring of Chap.6, add. content for 6.6, small corrections
27.07.2007	0.33	Katharina Wolter (UH)	Added Summary
27.07.2007	0.34	Katharina Wolter (UH)	Small additions in Summary and Sec. 6.6 corrections in Chap. 4+5
28.07.2007	0.35	Katharina Wolter (UH)	Completing Sec. 6.6, improvements and additions in Chap. 4, 5, Conclusion and Sec. 6.2, added section about WordNet
29.07.2007	0.36	Katharina Wolter (UH)	Completed figure captions, additions in Sec. 6.1, figure improvements
30.07.2007	0.37	Thorsten Krebs (UH)	Unification of terms
30.07.2007	0.38	Katharina Wolter (UH)	Corrections & improvements in whole document
31.07.2007	0.38	Katharina Wolter (UH)	Improved figures
31.07.2007	1.00	Katharina Wolter (UH)	Finalisation

Summary

The ReDSeeDS project is about requirements-driven software development. The major goal is to enable *reuse on the basis of requirements specifications*. One pre-condition for reaching this goal is the *retrieval* of requirements specifications that can be reused in the current software development project. The *software case similarity measure* described in this deliverable enables this retrieval.

The requirements engineer defines in a query the characteristics of software artefacts that might be reused in the current project. These artefacts are specified using the software case language (SCL) and stored in a *fact repository*. The software case similarity measure determines all software cases stored in the fact repository that are similar to a given query.

Conventional requirements specifications are often written in natural language. For such documents a similarity measure would usually only rely on lexicographic term matching. Due to the ambiguity of natural language this has significant disadvantages. Similar requirements specifications are only retrieved if the same words are used in both specifications. However, the same meaning can be expressed using different words, e.g. synonyms may be used. Furthermore, the same words may have a different meaning in some requirements specifications. This leads to the retrieval of requirements specifications that are not relevant for the current situation.

In ReDSeeDS, requirements specifications are written using the Requirements Specification Language (RSL). Taking advantage of the features of RSL the similarity measure can be more effective. RSL provides constrained and unconstrained language facilities for requirement description as well as conceptual modelling in combination with thesaurus features. The ability to map words to a common terminology solves the ambiguity problem of natural language.

In order to use the features of RSL to full capacity the similarity measure combines approaches from different research areas. The parts of specifications containing unconstraint language can be evaluated using measures from the field of Information Retrieval. Conceptual models can be compared more effectively using approaches from other research domains, like e.g. structural

Case-Based Reasoning, Description Logic and Graph-based similarity. Specific algorithms for determining the similarity of two terms contained in a conceptual hierarchy or thesaurus can be applied.

Our concept is based on a detailed definition of the requirements on the similarity measure. The approaches combined in the similarity measure are introduced in detail and compared regarding their applicability in RSL requirements specifications. The similarity measure described in this deliverable provides a conceptual framework that will be implemented during Work Package 5 (Development of the ReDSeeDSD system prototype). The effectiveness and efficiency of the described similarity measure will be evaluated in experiments.

Table of contents

Chapter 1

Scope, Conventions, Guidelines

1.1 Document Scope

This document provides a concept for the ReDSeeDS similarity measure that will be implemented in the ReDSeeDS Engine. The similarity measure provides the functionality of retrieving software cases that are similar to a particular query. This query is specified in the Software Case Query Language (SCQL). The software cases are written in the software case language (SCL) and are stored in the fact repository.

The conceptual idea of the ReDSeeDS similarity measure is based on a detailed definition of requirements to be met by the measure. The measure combines similarity approaches from different research areas in order to cope with the different aspects of requirements models. Graph-based similarity and Description Logic are used to compare the structural parts of requirements models. Similarity measures from Information Retrieval and Case-based Reasoning are used to compare the textual parts of requirements models.

1.2 Related work and relations to other documents

The results described in this deliverable are based on the work from different research areas. Main results from these research areas and related work are introduced in respective sections in Chapter 3.

Since this deliverable describes the similarity measure for ReDSeeDS software cases, it is based on the software case language (SCL) - the language used for representing software cases. This language is described in the following deliverables:

- D2.4.1: Requirements Specification Language Definition,
- D3.1: Software Case Meta-Model Definition,
- D3.2.1: Reuse-oriented Modelling and Transformation Language.

The similarity measure is mainly based on the requirements model contained in a software case. Thus, D2.4.1 is the most important input for this deliverable.

The RSL metamodel, which was defined in D2.4.1 has been adapted during Task T3.2. These modifications did not change the language from a user's perspective. However, for the similarity measure these modifications are relevant since the similarity measure defines which RSL classes need to be compared. For this reason the latest version of the RSL metamodel (so called *ToolReadyRSL*) was another important input for this task (see Deliverable D3.2.1).

The similarity measure described in this document is closely connected with the Software Case Query Language defined in Deliverable D4.1 (Software Case Query Language Definition) since the similarity measure is one main basis for the query language next to the software cases stored in the fact repository. Some of the similarity measure requirements (see Chapter 5) have implications for the query language.

The results of this deliverable provide input for Work Package 5, mainly Task T5.2 (Specification of user requirements for the ReDSeeDS Engine prototype) and Task T5.5 (Implementing the ReDSeeDS Engine prototype - Second iteration).

1.3 Document Structure

This deliverable is structured as follows: Chapter 2 illustrates the context in which the similarity measure is to be integrated and introduces important terms that are used throughout this deliverable.

Chapter 3 provides a brief overview of similarity definitions form different research areas. The described similarity approaches compare different types of artefacts and provide a similarity value. Some approaches also provide additional information, for example which parts of the artefact are similar and which parts are different.

Chapter 4 introduces the structure of RSL requirements models as background knowledge for the following chapters. This introduction concentrates on RSL entities relevant for the similarity measure.

Chapter 5 describes the requirements for the similarity measures in detail.

Chapter 6 presents a concept for the ReDSeeDS similarity measure. The chapter states how the similarity approaches described in Chapter 3 may be applied in order to compute the similarity of ReDSeeDS software cases.

Chapter 7 describes aspects that need to be considered when implementing the similarity measures.

Finally, Chapter 8 provides a summary of the main results.

1.4 Usage Guidelines

This deliverable should be used as a book that guides the reader through the basic similarity approaches described in literature, the requirements the software case similarity measure in ReDSeeDS must meet and the concepts behind this similarity measure. Thus, this deliverable mainly addresses Work Package 5 members who will integrate the requirements on the similarity measure in D5.2 (ReDSeeDS Prototype User Requirements Specification) and will implement them in Task 5.5 (Implementing the ReDSeeDS Engine Prototype - Second iteration). Since this deliverable mainly addresses ReDSeeDS members it is expected that the reader is familiar with the basic concepts of the software case language (SCL). Nevertheless, Chapter 4 provides a very brief overview of the major elements of the requirements specification language (RSL). This is reasonable due to the adaptations of the RSL metamodel in the tool-ready version.

Readers that are familiar with one of the similarity approaches introduced in Chapter 3 may skip the related section.

1.5 Conventions

The following notation conventions are used in this deliverable:

- sans-serif font is used for names of classes, attributes and associations, e.g. Requirement
- if a class name is used in description of package other than the one it is included in, it is preceded with package name and a double colon ("::"), e.g. RequirementsSpecifications::Requirement
- ***bold/italics font*** is used for emphasised text, e.g. ***Abstract syntax***
- `typewriter font` is used for examples, e.g. `Customer`

Chapter 2

Introduction

The main objective of Workpackage 4 is developing technologies that enable the reuse of cases. This deliverable provides the background for implementing a *similarity measure* for seeking similar cases and is thus a major input for Task 5.4 & 5.5 (Implementing the ReDSeeDS Engine prototype).

A *software case* in the context of ReDSeeDS is a set of artefacts that are produced during the development of a software product together with all their interconnections. A software case contains a requirements model, an architectural model, several subsystem models, the code and transformational information, and potentially other development artefacts. All these artefacts shall be described using the ReDSeeDS Software Case Specification Language (SCL).

Software cases can vary in size. Firstly, because software products vary in size and secondly, because the artefacts that constitute a software case can also describe parts of a software product only. The requirements model of one software case may describe only one specific aspect of a software product while another requirements model specifies an entire product.

The part of SCL that describes requirements is called Requirements Specification Language (RSL). Since the similarity measure mainly evaluates the requirements models of software cases, RSL is of special importance for this deliverable. The requirements part of software cases consists of a requirements specification and a domain specification, together called *requirements model*. The requirements specification consists of requirements and requirements representations. The domain specification describes main entities of the application domain using a conceptual model and phrases. The conceptual model represents entities form the application domain using object-oriented principles (generalisation, aggregation, association with multiplicities) while the phrases offer textual information similar to a software case specific glossary.

The *fact repository* contains all information about the artefacts that constitute the software cases. Additionally, the fact repository contains general knowledge e.g. the *Terminology*, an overall "dictionary" on which all domain specifications are mapped. Figure 2.1 shows the fact repository with software cases of different size and their relation to the general knowledge.

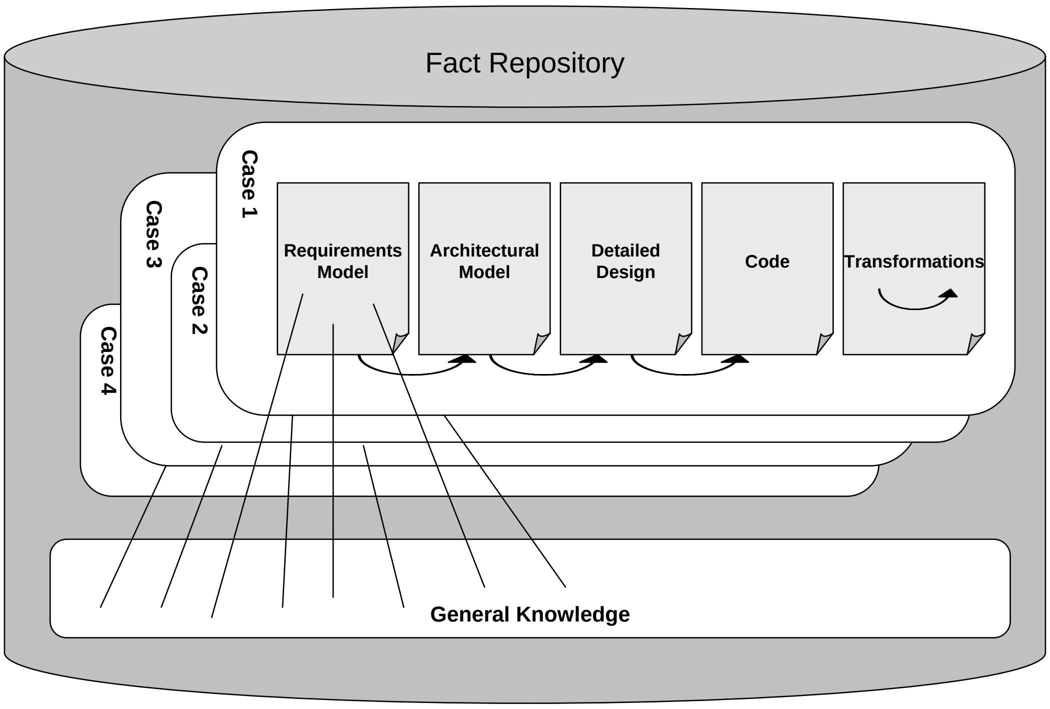

Figure 2.1: The fact repository consists of software cases and general knowledge.

Chapter 3

Existing Similarity Measures

Measures that capture similarity of artefacts have been developed in many research communities and for different types of artefacts. In the following we briefly introduce some of these approaches. This introduction is restricted to approaches that are relevant for comparing RSL requirements models.

RSL requirements models can contain different types of requirements representations: textual representation (natural and controlled language), graphical representations and even images (screenshots). The Similarity of textual documents to a given query is the main topic of *Information Retrieval* (see Section 3.1). Some Information Retrieval approaches apply thesauri or semantic lexicons. Section 3.2 introduces *WordNet*, a semantic lexicon that is based on the concept of synonym sets that group synonymic words. *Case-based reasoning* (CBR) examines the similarity of so-called cases that store past experiences. Different case representations are distinguished [BBMW99]. For RSL specifications *Textual CBR* and *Structural CBR* are relevant (see Section 3.3). Conceptual representation of queries and cases may be addressed with a description-logic based approach. Section 3.4 describes how knowledge representation mechanisms known from description logics can help in defining queries and measuring similarity between a query and cases. Since graph-based representations play an important role in RSL, *similarity of graphs* is introduced in Section 3.5.

RSL also contains elements that contain visual information like screenshots or icons. Identifying screenshots with similar structure of user interface elements would require content-based image retrieval or even image understanding. Whether such approaches would actually improve the similarity measure for software cases is questionable. Since they are out of scope of ReDSeeDS they are not discussed in the following.

3.1 Similarity in Information Retrieval

Information Retrieval (*IR*) addresses the search of relevant documents [GF04] in (large) document collections. Some views also include the representation, storage and organization of information items [BYRN99, SM83]. A major application domain of IR methods are web search engines.

Information Retrieval distinguishes mainly four types of models: boolean [GF04], vector space [GF04], probabilistic [GF04], and inference network models [GF04]. The boolean models offer a pure matching without any ranking by relevance / similarity. The three most used models in IR research are the other three models [Sin01].

IR system are typically installed for searching in large document collections. This has the consequence that indexing is done offline [SM83]. Thus, those documents not yet indexed will not be found by the retrieval. This behavior might conflict with the expectations of a user who expects that documents stored in a system are immediately available for retrieval.

Similarity in the different IR models is determined as follows [BYRN99, Sin01]:

Boolean models Pure matching without relevance / similarity ranking.

Vector space models Each document and query is represented as a vector of terms. The similarity (also called document score) is calculated between a query vector and a document vector, which is usually a value between 0 (no similarity) and 1 (identical). For each document vector and term, a document-specific weight is assigned to a term (see *term weighting* below). The document score is then used for ranking the documents.

Probabilistic models These models estimate the probability of the relevance of the documents for a query. This probability is then used for ranking the documents.

Inference network model The document retrieval is modeled as an inference process in an inference network. In its simplest implementation, a document instantiates a term with a certain strength, and the credit from multiple terms is accumulated given a query to compute the equivalent of a numeric score for the document.

For the so-called *term weighting*, the following measures are usually combined: term frequency (number of occurrences of term in document), document frequency (number of documents that contain the term) and document length. The state of the art in term weighting are the Okapi weighting and the pivoted normalization weighting [Sin01]. However, real differences are only experienced with large collections (i.e., > several 10,000 documents). Thus, some "basic" IR methods should do for ReDSeeDS.

Terms that occur too often as well as rare terms are considered as useless for distinguishing between documents and are therefore usually ignored in IR models / techniques. However, for a system like the ReDSeeDS engine, this could become a critical issue: If a word from the domain vocabulary was not part of the index, relevant software cases would not be found if this word were a part of the query.

The effectiveness of Information Retrieval methods is usually evaluated using the following two measures [GF04]:

Recall Ratio of the number of relevant documents retrieved to the total number of relevant documents.

Precision Ratio of the number of relevant documents retrieved to the total number retrieved.

[LBSBW98] mentions two problems not solved by traditional IR methods:

Ambiguity Problem The term vectors representing two documents may be very similar but the actual meaning of the documents is not.

Paraphrase Problem The vector of terms representing two documents is not similar but the actual meaning is.

The authors stress the importance of domain knowledge to solve these problems. Examples for such domain knowledge are thesauri or a sematic lexicon as discussed in the next section.

3.2 WordNet

WordNet[1] is a semantic lexicon that was developed at the Cognitive Science Laboratory at Princeton University [MBF$^+$90]. WordNet is based on the concept of synonym sets (called *synsets*) that group synonymic words or sequences of words (like "computer mouse") of the English language. WordNet contains nouns, verbs, adjectives and adverbs. Amongst others

[1] see: http://wordnet.princeton.edu/

the following semantic relations connect synsets: *hypernyms* / *hyponyms* (is-a, is-a invers) and *holonym* / *meronym* (part-of, part-of invers). For each synset WordNet provides definitions or example sentences. Words with different meanings participate in several synsets. For such words a *frequency score* describes which meanings are more common and which are used rather infrequently.

WordNet is freely available for download and can also be accessed with any Web Browser[2]. Several APIs are available for download (Java, .NET etc.) and several similarity measures for synset pairs have been published based on WordNet. [PPM04] gives an overview of such measures, which are all freely available as Perl packages.

How WordNet can be applied for solving the paraphrase and ambiguity problem is a matter of current research [VVR$^+$05, LYM05]. [Kur04] describes in detail how the vector space model described in Section 3.1 can be extended by application of WordNet or other semantic lexicons.

WordNet has been applied in combination with Case-based Reasoning approaches (see next section) for retrieval of UML class diagrams [Gom04, GPP$^+$04].

3.3 Similarity in Case-based Reasoning

Case-based reasoning (*CBR*) [Kol93, AP94, BBG$^+$99] is the process of solving new problems based on the solutions of similar past problems. A car mechanic who fixes an engine by recalling another car that exhibited similar symptoms is using case-based reasoning. A lawyer who advocates a particular outcome in a trial based on legal precedents or a judge who creates case law is also using case-based reasoning. And a software engineer who is reusing existing software artifacts is also using case-based reasoning.

It has been argued that case-based reasoning is not only a computer-based reasoning method, but also a pervasive behavior in everyday human problem solving. CBR can be brought to an organizational level and then used as a principle for knowledge management [TA98, TA97]. How to link recording and reusing experiences to business processes is subject to the field of *Experience Management* [NAB07].

Application areas of case-based reasoning are broad: classification, diagnosis, and decision support tasks as well as planning, configuration, and design tasks [ABS96]. With this, CBR

[2]see: http://wordnet.princeton.edu/perl/webwn

has been used for product recommendation, fault diagnosis of technical systems, help desks, medical diagnosis support, medical treatment planning support, software reuse, etc.

Case-Based Reasoning (CBR) distinguishes three different approaches [BBMW99]: Structured CBR, Textual CBR, Conversational CBR.

- The Structured CBR approach uses attribute-value pair representations. Such cases can be found in well-structured domains. Similarity measures are specified per attribute (local similarity) and as a "‘summarizing"’ global similarity measure that combines the local similarities. More advanced representations are object-oriented (e.g, REFSENO [TG98] and CBR-Works (tecinno / empolis GmbH)). However, Nick [Nic05] has shown that for context-based retrieval as required for software artifacts [BR91], a flat attribute-value similarity model with multiple views is sufficient for a context-based retrieval on software artifacts.
- The Textual CBR approach runs on textual documents rather than structured cases. From a pure users’ view, this is very similar to Information-Retrieval approaches. However, internally, Textual CBR makes use of domain-specific vocabulary and similarity measures [LHK98], while Information Retrieval builds upon general heuristics and statistics for this purpose.
- Conversational CBR has mainly been used for call-center applications. It represents cases as trees of question-answer pairs. For these reasons, it is considered irrelevant for ReDSeeDS.

Today, Structured CBR and Textual CBR are combined in one approach (e.g., CBR-based tool *e:IAS Enterprise* from empolis[3]). This allows us to give some structure to documents and let local similarity measures benefit from Textual CBR similarity measures. Similarity measures in Case-Based Reasoning are usually normalized, i.e., a similarity is a value $\in [0..1]$ where 1 means a full / exact match and 0 means no similarity.

In general, any similarity for a finite set of values can be specified using a *value range* $\times$ *value range* table. However, for data types with larger value ranges, this is obviously not really practical. Thus, CBR research has developed a number of (local) similarity measures for the following datatypes in Structured CBR.

Typical local similarity measures in Structured CBR exist for: Boolean, Integer, Real, String / Symbol. For Symbols, there is also a taxonomic similarity [Ber98, TG98].

[3]www.empolis.com

Also at the level of local similarity measures are similarity measures associated with type constructors: Set and Interval [TG98].

Typical global similarity measures - for combining these local similarity measures - are

- weighted normalized sum

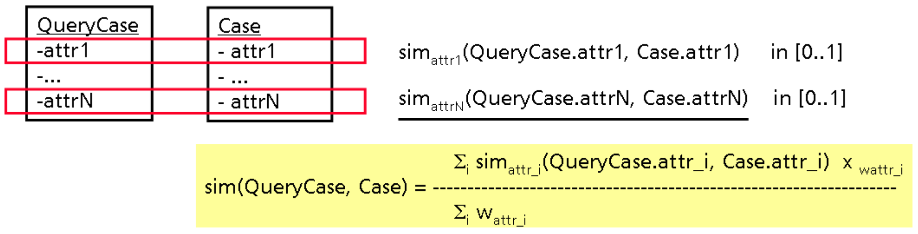
QueryCase
-attr1
-...
-attrN
Case
- attr1
- ...
- attrN
sim_attr1(QueryCase.attr1, Case.attr1) in [0..1]
sim_attrN(QueryCase.attrN, Case.attrN) in [0..1]
sim(QueryCase, Case) = Σ_i sim_attr_i(QueryCase.attr_i, Case.attr_i) x w_attr_i / Σ_i w_attr_i

Figure 3.1: Similarity in Case-Based Reasoning

- euclidic sum

The euclidic sum calculates the euclidic distance in a mathematical / geometrical sense. However, this requires numeric attributes. Thus, the distance measure is usually generalized as a weighted normalized sum. Since ReDSeeDS' attributes are mainly text / word-based, the weighted normalized sum is the choice for ReDSeeDS.

For `consists-of` or `part-of` hierarchies, e.g., in object-oriented representations, global similarity measures can be combined: global similarities for parts are local similarities for the compound structure. For a normalized weighted sum, this could obviously be easily mapped to a flat representation.

Nick [Nic05] described how to implement context-based retrieval with a standard Structured CBR system.

3.4 Similarity in Description Logic combined with Case-based Reasoning

Description Logics (*DLs*) (also known as terminological knowledge representation systems or concept languages) have been developed from semantic networks [Qui69], frame-based languages [Min74] and the concept language KL-ONE [BS85]. In general, the knowledge base constitutes a logical theory and description logic systems allow us to compute whether a model of the domain concerned exists. Such knowledge representation is well-suited for reasoning about terminological knowledge, ontologies and configuration of technical systems. One of the best-known applications of description logics is the *Semantic Web*.

In the following we give a brief introduction of description logics and their knowledge representation facilities. After that we discuss computing similarity measures in DL systems. For a detailed introduction of description logics we refer the interested reader to, for example, [BCM$^+$03].

Introduction

Basic representation facilities are *concepts* and *roles*. Concepts gather common properties of a set of objects and roles are binary relations on concepts. *Terminological knowledge* (*TBox*) and *assertional knowledge* (*ABox*) are distinguished. The terminological knowledge represents a terminology of the relevant domain by using concepts while the assertional knowledge represents concrete situations as *instances* (also called *individuals*). Complex knowledge structures can be constructed by combining concepts and roles using conjunction, disjunction and negation, by using existential and universal quantification, value restrictions and number restrictions or by defining inverse roles, transitive roles and concrete domains.

To make clearer the inferences a DL system offers, let us consider a classic example from the description logic literature: the terminological knowledge contains the concepts `human`, `man`, `male` and `female`. The concept `human` specifies a role `gender` with possible fillers `male` and `female`. The concept `man` specializes the `gender` role with the filler `male`. The assertional knowledge contains `John` as an instance of the concept `human`.

Basic inference services of description logics systems are listed in the following:

Automatic computation of concept subsumption : The description logic system automatically infers a concept's position in the taxonomy based on the roles this concept specifies. The description logic system infers that the `human` concept subsumes the `man` concept (`man` $\subseteq$ `human`).

Consistency checking of a concept with regard to the TBox : The description logic system can automatically detect if a concept definition is inconsistent with the rest of the terminological knowledge. A concept is assumed to be consistent when it allows the creation of at least one instance. For example, it is not allowed to specify the same role twice with different fillers.

Consistency checking of the ABox with regard to the TBox : The description logic system can automatically detect whether assertional knowledge is consistent with the terminological knowledge on which it is based. Any instance in the ABox is an instance of a concept definition in the TBox. For example, it is not allowed for any instance of concept `human` that defines the `gender` role to specify a filler other than `male` or `female`.

Deduction of new facts from existing facts : `John` is defined as an instance of the concept `human`. `John` additionally specifies a role `gender` with filler `male`. The DL system now automatically infers that `John` is not only a `human` but also a `man`.

3.4.1 Combining Description Logics with Case-based Reasoning

Case Representation

The fact repository contains all cases represented by concepts in the TBox. This means that every software case of the fact repository is represented by a concept. The concept `software case` specifies the roles `requirements`, `architecture`, `design`, `code` and `transformations`. The fillers of these roles are concepts representing the RSL and SCL classes, respectively. When a new software case is stored in the fact repository, a new concept, for example `Case-1`, is created in the TBox and the requirements, architecture, design and code are represented by a set of concepts that are specified as fillers in the corresponding roles of the `Case-1` concept.

A query is specified as an instance of the `case` concept, potentially containing a set of roles defining the requirements, architecture, design and code. The DL system automatically recognises the most specific concept of which the `case` instance is an instance. Similar cases can be easily recognised now because they are located close to each other in the taxonomy.

When a new case is stored in the fact repository, a new `case` concept is created in the TBox. The distinction between the fact repository and the case that represents the current development is thus reflected by distinguishing the fact repository, which is represented in the terminological knowledge, and the currently developed case, which is represented in the assertional knowledge. This view of knowledge representation is quite natural since concepts describe objects from which multiple instances can be created: this means that cases from the fact repository describe cases that can be retrieved and instantiated for future development.

Similarity

Computing the similarity of two concept definitions in description logic systems involves the conceptual structure, that is both concept placement in the taxonomy and roles that a concept defines. We distinguish between *distance-based similarity* and *role-based similarity* [GCGADA99]:

Distance-based similarity Exploiting the taxonomic structure, a typical way to evaluate similarity of two concepts is to determine the distance between the concepts in the taxonomic structure. The semantic interpretation of the distance is: the shorter the path between the two compared concepts is the more similar they are. This *intensional* approach relies on the assumption that links in the taxonomy represent uniform distances. Another possibility to compute similarity of two concepts is the *extensional* approach: comparing the number of their common instances. The semantic interpretation of the number of common instances is: the more common instances both concepts share, the more similar they are. This approach relies on the assumption that all concepts in the taxonomy are evenly populated with instances.

Role-based similarity The basic idea of role-based similarity is that two concepts are more similar when their subgraphs, based on specified roles, are more similar. When comparing roles their most important aspect is their fillers: concepts or concrete domains that specify the value of a role. When comparing two concepts, the function of role-based similarity is recursively applied. The recursion terminates when comparing concepts without roles, whose similarity is given by the concept-based similarity function. Cyclic roles have to be taken into account during implementation. When comparing two concrete domains, standard mechanisms known from mathematics can be used. Concrete domains that are typically used in description logics are strings, integer and real numbers, sets of strings, integers and reals and ranges of integers and reals. Similarity of concrete domains can be measured by computing the distance between numbers and intersections of sets or ranges.

The similarity between two concept definitions is the sum of distance-based similarity and role-based similarity. A value between 0 and 1 is computed, 0 denoting no similarity and 1 denoting equality. A detailed description of the similarity measure including algorithms for computing similarity of concepts is given in Section 6.4.

3.4.2 Example

Let us consider a simplified RSL-related example: two SVOSentences need to be compared. When comparing SVOSentences, the structure of the sentences is known (i.e. Subject and Predicate) and all Terms of a sentence are compared to the appropriate Terms of the other sentence. All Terms that are used in a sentence are defined in the DomainSpecification and have an association to the corresponding Terms defined in the Terminology. Definitions of the DomainSpecification are represented by concepts and their associations are represented by roles, the concepts representing Terms being the fillers of these roles. All these concepts are modeled in the TBox.

When, for example, a new SVOSentence contains the Term `order`, a concept instance representing this Term is created (in the ABox). This instance contains a role with the concept representing the Term `order` in the Terminology as its filler. If a concept representing the Term `order` is already modeled in the TBox, it is automatically inferred that the new instance is an instance of the pre-existing concept. The similarity of this concept and the newly created instance is 1 since they both have the same role with the same filler.

A concept representing an SVOSentence contains roles specifying the NounPhrases and VerbPhrases. Hence, comparing SVOSentences is done analogously to comparing definitions from the DomainSpecification. In fact, these definitions from the DomainSpecification are compared in the first recursion of comparing SVOSentences.

3.5 Graph-based Similarity

Beside the similarity approaches depicted in the previous sections, *graph-based similarity* will play an important role in the ReDSeeDS similarity measure. In this deliverable and the whole ReDSeeDS project, graph-based similarity includes all attempts to define and measure the similarity of two artefacts by their graph representation. In ReDSeeDs, these graph-representations will be abstract syntax graphs stored in the fact repository.

3.5.1 General principles

The similarity of two artefacts represented as graphs is defined mainly by two properties of the graphs. The first one is the local similarity of single parts in the graphs, represented as nodes and edges. The second one is the structure of the graphs, defined by connections of nodes and edges.

Currently, there are various definitions of graph-based similarity or graph-based differences available. Beside the exact match as maximum similarity of two graphs, there are mainly the following three ideas used as a basis for graph similarity algorithms.

- Structural similarity based on (sub)graph isomorphism
- Difference between two graphs depicted as the number of editing operations needed to convert one graph into the other
- A combination of structural and local similarity, also known as the similarity flooding idea

The main principles of the three approaches are depicted in the following subsections. The third one as the most promising one is described in more detail.

3.5.2 Structural similarity

The first of these three ideas, the structural similarity based on (sub)graph isomorphism, tries to find the largest subgraphs in two graphs *A* and *B* that are isomorphic. Based on the size and number of these isomorphioc subgraphs, the similarity of the two graphs *A* and *B* is defined. As an additional factor, the complexity of the isomorphism can be used to calculate the similarity of two graphs.

The result of a similarity measurement based on (sub)graph isomorphism is a real value that expresses how similar two graphs are. Further, the subgraphs that are isomorphic can be treated as a result of the algorithm.

The main restriction of subgraph isomorphism based similarity lies in the isomorphism. Only isomorphic parts are matched, while parts the are only similar cannot be matched. Thus, subgraph isomorphim similarity may be only a starting point for a graph-based similarity approach in ReDSeeDS.

3.5.3 Local similarity

The second similarity measure depends on the idea that the similarity of two artefacts is the converse of their local differences. The difference between two artefacts can be seen as the minimal number of editing operations needed to convert one artefact into the other. As each of these editing operations affects only a single node or edge in the graph, the similarity measure is based on local differences or local similarities.

This approach has been used in text-based similarity since the nineteenseventies, for instance by the diff-utilities [Hun76]. Experiments to apply difference measures for text on textual representation of graphs, for instance XMI, show that the calculated differences do not match the differences expected by humans.

The result of such a difference or similarity measure is a list of editing operations needed to transform one document into the other one. Using the number and complexity of that operations as well as the size of the documents as a basis, a real value can be calculated which expresses the similarity of two documents.

Further research shows, that the algorithms known from difference calculation for textual documents produce poor results if used to calculate differences between graphs. Thus, the difference-based similarity may also be only a starting point for a graph based similarity approach in ReDSeeDS.

3.5.4 A combined approach

The third similarity measure mentioned above combines structural similarity of graphs and local similarity of single elements. This idea is known as similarity flooding [MGMR02]. The main concept of this approach is the fact, that local similarity and structural similarity highly depend on each other. Intuitively, the local similarity of two graph elements is mainly defined by their attributes. As an example, the similarity of two string attributes can be expressed by the longest common substring or, a bit more sophisticated, by the number of edit operations needed to transform one string into the other one. The similarity of other attributes can be calculated in similar manner.

Further, also connections to other graph elements affect the local similarity of two graph elements. The second part of the overall similarity is the similarity of the graph structure. This graph structure is defined by interconnections of graph elements and therefore highly influenced by the local similarity of the several graph elements. Thus, the similarity flooding can somehow be seen as a combination of subgraph-isomorphism and editing difference approaches. The result of such a graph similarity measure is a set of matches between elements in both models. Every element in the first model has at most one matching element in the second model and vice versa. Additionally, every match is weighted with a factor between 0 (no match) and 1 (full match). On the basis of the matches together with the match factors and the elements that have no matching partner, the total similarity of two models can be calculated.

To accommodate the facts described above, several algorithms were developed during the last years. The main idea behind these algorithms is to calculate the local similarity of one node N and to propagate this similarity to all connected nodes. The local similarity of these nodes connected to N is then influenced by the local similarity on N. Of course, the local similarities of the nodes connected to N will influence the local similarity of N itself. So, the similarity calculation is done in an iterative process until the similarities do not change any more.

One of the variants which is known to work even for large graphs is called *SiDiff*[4] [UK05]. It is developed by the software engineering group at the University of Siegen, Germany. The SiDiff similarity algorithm can be used to calculate similarities of software case artefacts by running the SiDiff algorithm on the data stored in the software case fact repository. Details of this coupling are a matter of current research.

The application of graph-based similarity measures and computation is mainly useful on well-structured data. In software cases, this well-structured data are SVOSentences, ConstrainedLanguageScenarios and model-based Scenario representations as well as UML diagrams. The quality of graph-based similarity highly depends on the right adjustment of several parameters in the SiDiff algorithm and thus on experiments with real software cases. These experiments are subject of current research.

[4]www.sidiff.org

Chapter 4

RSL Requirements Models

This chapter provides a brief overview of the structure of requirements models specified in RSL and their relations to the Terminology. This introduction only presents aspects of RSL requirements models that are important for the similarity measure. For a detailed introduction of RSL we refer to Deliverable D2.4.1 (Requirements Specification Language Definition). Since this section is based on the revised RSL metamodel some details differ from D2.4.1, however.

Some RSL aspects already imply requirements for the similarity measure. Whenever this applies the number of the requirement is mentioned. All requirements on the similarity measure are described in detail in the next chapter.

4.1 Requirements Specification

An RSL requirements model consists of a RequirementsSpecification and a DomainSpecification. This section introduces the RequirementsSpecification while the DomainSpecification is described in Section 4.6.

The RequirementsSpecification is organised in several RequirementsPackages which contain the Requirements. This structure is illustrated in Figure 4.1. RSL distinguishes five types of Requirements:

- FunctionalRequirementOnSystem
- FunctionalRequirementOnComposite

- ConstraintOnSystem
- ConstraintOnProcess
- UseCase

Requirements can be associated with other Requirements using RequirementRelationships like Fulfills, Operationalises, Constrains, MakesPossible. Furthermore, each Requirement has one or more RequirementRepresentations. RSL provides the following types of RequirementRepresentations:

- DescriptiveRequirementRepresentations:
  - SentenceList
  - ConstraintLanguageScenario
- Model-basedRequirementRepresentations:
  - ActivityScenario
  - InteractionScenario

In order to find similar Requirements it is necessary to compare Requirements having different representations. Some specific Requirement may be specified in a SentenceList in one software case an in a ConstraintLanguageScenario in another software case (see Requirement 6).

4.2 Sentence List

A SentenceList consists of one or more HyperlinkedSentences. HyperlinkedSentence is an abstract class that can be specialised into the following classes:

- NaturalLanguageHypertextSentence
- ConstraintLanguageSentence

ConstraintLanguageSentence is also an abstract class. A SentenceList may contain the following concrete types of Sentences:

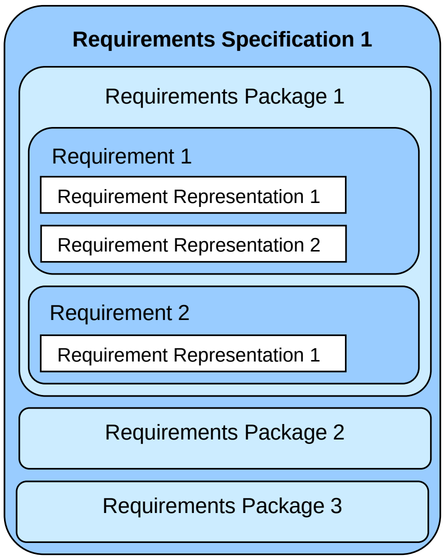
Requirements Specification 1
Requirements Package 1
Requirement 1
Requirement Representation 1
Requirement Representation 2
Requirement 2
Requirement Representation 1
Requirements Package 2
Requirements Package 3

Figure 4.1: Requirements are organised in RequirementsPackages and have one or more RequirementRepresentations.

- NaturalLanguageHypertextSentence
- SVOSentence
- ModalSVOSentence
- ConditionalSentence

```
1. [[d:The n:system]] a: shall allow [[d:the n:shop assistant]] [[v:to define m:new n:courses]].
2. [[d:The n:system]] a: shall [v:display d:all n:courses]].
3. [[d:The n:system]] a: shall allow [[d:the n:shop assistant]] [[v:to cancel n:courses]].
4. [[d:The n:system]] a: shall allow [[d:the n:shop assistant]] [[v:to print n:course list]].
5. [[c:If]] [[d:the n:shop assistant]] [[v:cancels d:a n:course]],
   [[d:the n:system]] a:shall [[v:inform d:the n:course instructor]].
```

Figure 4.2: A SentenceList containing different types of HyperlinkedSentences.

An example for a SentenceList is given in Figure 4.2. This Figure illustrates that a SentenceList can contain different types of Sentences. The same aspect of a Requirement can be stated in an SVOSentence in one SentenceList and in a NaturalLanguageHypertextSentence in another SentenceList. Thus, in order to compare SentenceLists it might be necessary to compare all different Sentence types a SentenceList may contain, i.e.: NaturalLanguageHypertextSentence, SVOSentence, ModalSVOSentence, ConditionalSentence (see Requirement 4).

4.3 Constraint Language Scenario

ConstraintLanguageScenarios consist of a sequence of ConstraintLanguageSentences and a UIStoryboard. Figure 4.3 gives an example for the ConstraintLanguageSentence part of a scenario.

```
Precondition: Customer must be logged in
1. [[n: Customer]] [[v: wants to sign up p: for n: exercises]]
==> invoke/insert: Choose exercises type
2. [[n: FC System]] [[v: checks n: availability p: of n: exercises]]
==> cond: exercises unavailable
3. [[n: FC System]] [[v: shows n: error message dialog]]
final failure
```

Figure 4.3: A ConstraintLanguageScenario containing three SVOSentences and one InvocationSentence, one ConditionSentence, a PreconditionSentence and a PostconditionSentence. The UIStoryboard is not shown in the figure.

ConstraintLanguageSentence is an abstract class. As part of a ConstraintLanguageScenario it can be specialised into the following types of sentences:

- SVOSentence
- InvocationSentence
- PreconditionSentence
- PostconditionSentence
- ConditionSentence

UIStoryboard contains a sequence of UIScenes. The transition from one UIScene to the next one is initiated by a UserAction which is associated with an SVOSentence. A UIScene consists of a UIPresentationUnit (screenshot) and has two parameters: sceneNumber and sceneDescription. Thus, the ConstraintLanguageScenario is the only RequirementRepresentation that may contain Requirements information in images.

4.4 Activity Scenario

An ActivityScenario consists of RSLActivityEdges and RSLActivityNodes. An RSLActivityNode contains an SVOSentence or a ControlSentence. The RSLActivityEdge may contain a ConditionSentence and a RejoinSentence. An example for an ActivityScenario is given in Figure 4.4.

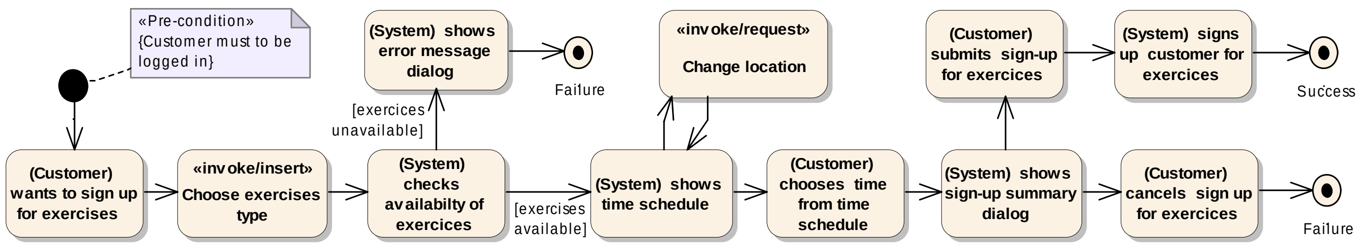
«Pre-condition»
{Customer must to be logged in}
(System) shows error message dialog
Failure
«invoke/request»
Change location
(Customer) submits sign-up for exercices
(System) signs up customer for exercices
Success
[exercices unavailable]
(Customer) wants to sign up for exercises
«invoke/insert»
Choose exercises type
(System) checks availabilty of exercices
[exercises available]
(System) shows time schedule
(Customer) chooses time from time schedule
(System) shows sign-up summary dialog
(Customer) cancels sign up for exercises
Failure

Figure 4.4: ActivityScenario

4.5 Interaction Scenario

An InteractionScenario is one possible representation for Scenarios which is similar to the interaction diagram in UML. It consists of Lifelines and Messages between the Lifelines. Each Lifeline represents an Actor or a SystemElement. Communication between Actors and SystemElements are modelled as Messages. Together with the Lifelines as subjects, the Messages are SVOSentences.

See Figure 4.5 for an example Interaction Scenario.

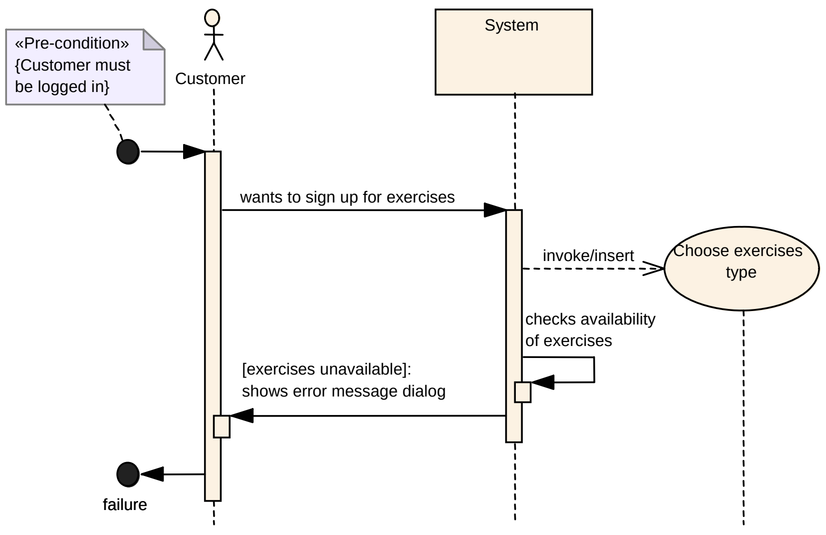
«Pre-condition»
{Customer must be logged in}
Customer
System
wants to sign up for exercises
invoke/insert
Choose exercises type
checks availability of exercises
[exercises unavailable]: shows error message dialog
failure

Figure 4.5: InteractionScenario

4.6 Domain Specification

The Domain Specification contains definitions for all Terms and Phrases referred to in the RequirementsSpecification and separates this representation of the application domain from the RequirementsSpecification. It can be seen as a precise, software-case-specific glossary. In contrast to DomainSpecifications the Terminology provides a definition of Terms that is common for all software cases. The elements of all DomainSpecifications are mapped to Terms of the Terminology. This mapping provides a word sense disambiguation for all words used in a DomainSpecification. Thus, the ambiguity problem (see 3.1) is solved for RSL requirements models. Figure 4.6 illustrates the relations between elements of the RequirementsSpecification, the DomainSpecifications and the Terminology. The Terminology is described in the next section.

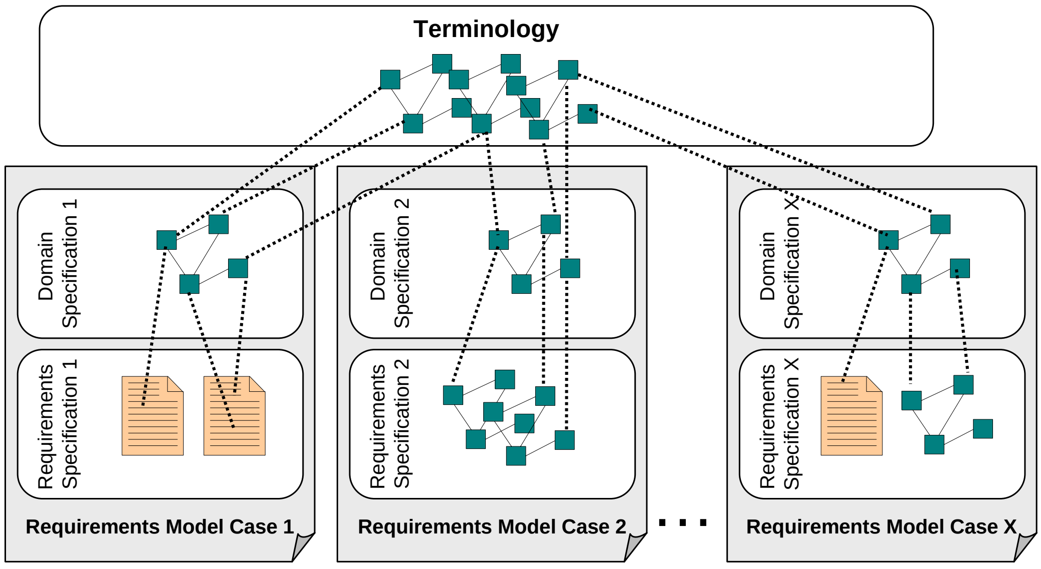
Terminology
Domain Specification 1
Requirements Specification 1
Requirements Model Case 1
Domain Specification 2
Requirements Specification 2
Requirements Model Case 2
...
Domain Specification X
Requirements Specification X
Requirements Model Case X

Figure 4.6: Mapping between RequirementsSpecification, DomainSpecification and Terminology.

The Domain Specification is structured in packages like NotionsPackage, ActorsPackage and SystemElementsPackage. Additional packages for further structuring can be defined. An example for such a package is the UIElementsPackage in Figure 4.7.

The NotionsPackage contains all Notions. Each Notion contains a Noun from the Terminology as name. A Notion groups all DomainStatements that share its name. A DomainStatement contains a Phrase as its name and NaturalLanguageHypertextSentence as description. Notions may be generalised.

UIElements, InputOutputDevices and InputOutputTyps are specialisations of Notion. They are used to describe UserInterfaceRequirements. Each UIElement has at least a UIElementRepresentation. This might either be a textual description or an image, screenshot, icon, voice etc.

The ActorsPackage contains all Actors and the SystemElementsPackage groups all SystemElements.

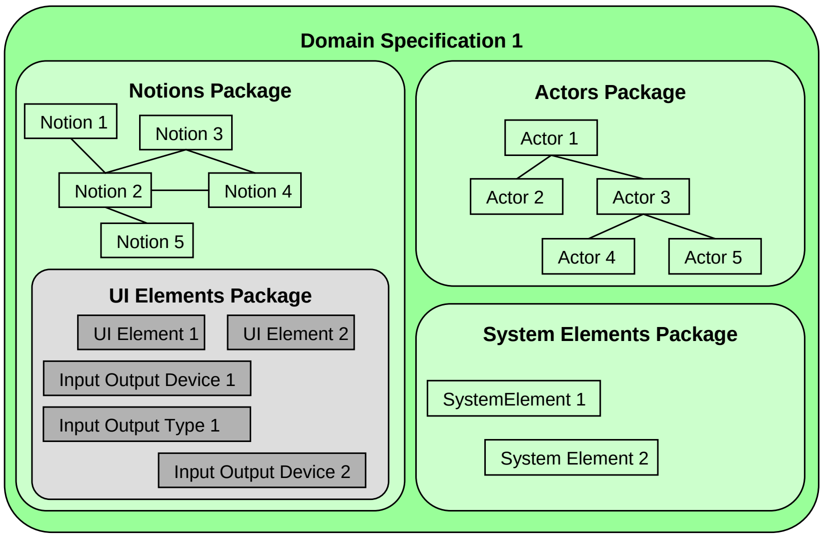

Figure 4.7: Example structure of a DomainSpecification

Figure 4.8 gives an example for a minimal RequirementsSpecification that contains only one SVOSentence and the associated DomainSpecification.

4.7 Terminology

The Terminology contains all words that are used in the RequirementsSpecification. For each word, the possible inflections are stored in the Terminology. Further, a short explanation of every word may be added. The different words in the Terminology are connected by links, so synonyms and homonyms can be identified as well as words that are “somehow similar”. In contrast to the DomainSpecification, the Terminology contains general knowledge onto which all DomainSpecifications are mapped. The relation between DomainSpecifications and Terminology is illustrated in Figure 4.9 on page 28.

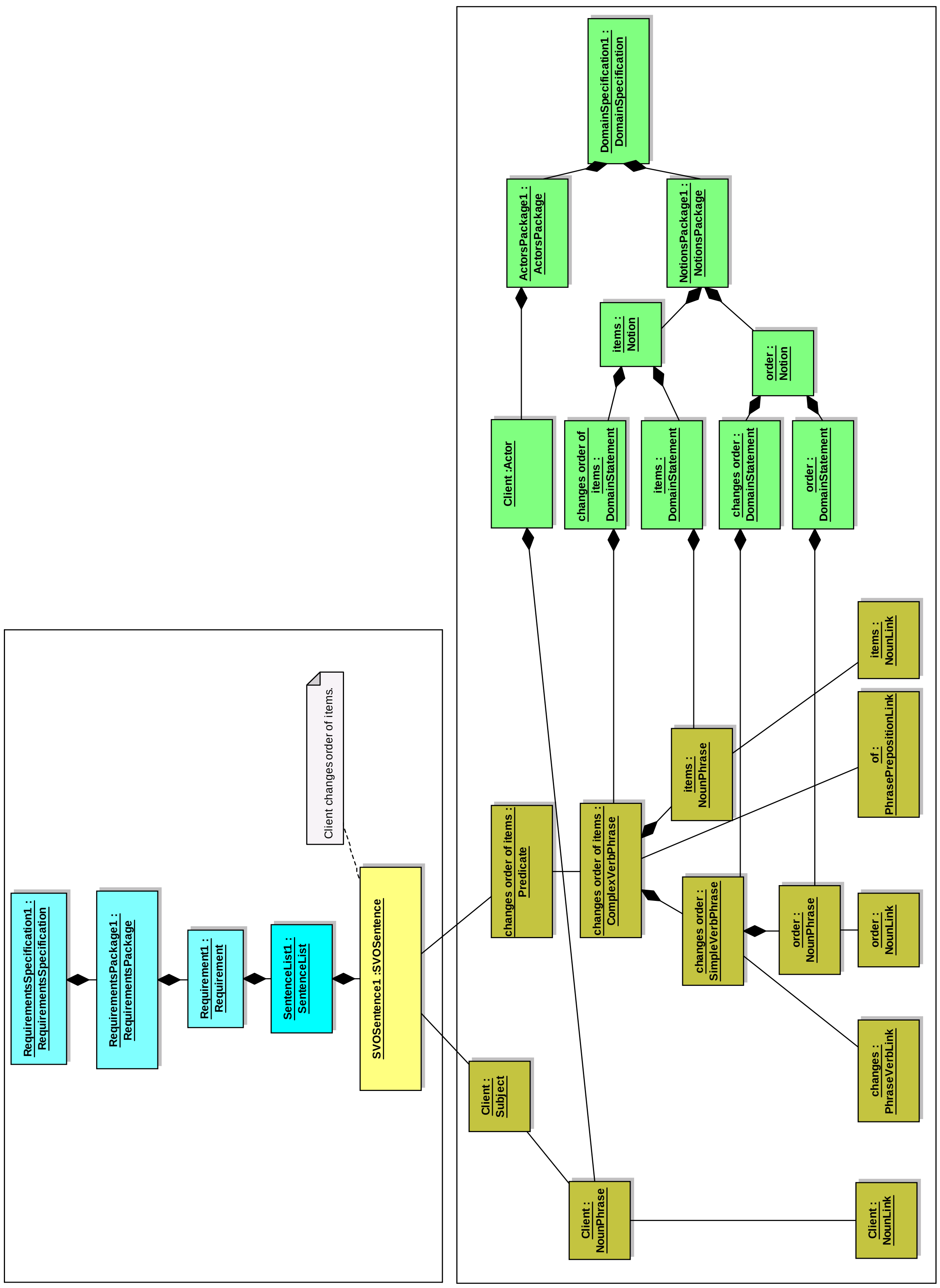
RequirementsSpecification1 : RequirementsSpecification
RequirementsPackage1 : RequirementsPackage
Requirement1 : Requirement
SentenceList1 : SentenceList
SVOSentence1 :SVOSentence
Client changes order of items.
Client : Subject
changes order of items : Predicate
Client : NounPhrase
changes order of items : ComplexVerbPhrase
items : NounPhrase
changes order : SimpleVerbPhrase
order : NounPhrase
Client : NounLink
changes : PhraseVerbLink
order : NounLink
of : PhrasePrepositionLink
items : NounLink
Client :Actor
changes order of items : DomainStatement
items : DomainStatement
changes order : DomainStatement
order : DomainStatement
items : Notion
order : Notion
ActorsPackage1 : ActorsPackage
NotionsPackage1 : NotionsPackage
DomainSpecification1 : DomainSpecification

Figure 4.8: RequirementsSpecification containing one SVOSentence and related DomainSpecification

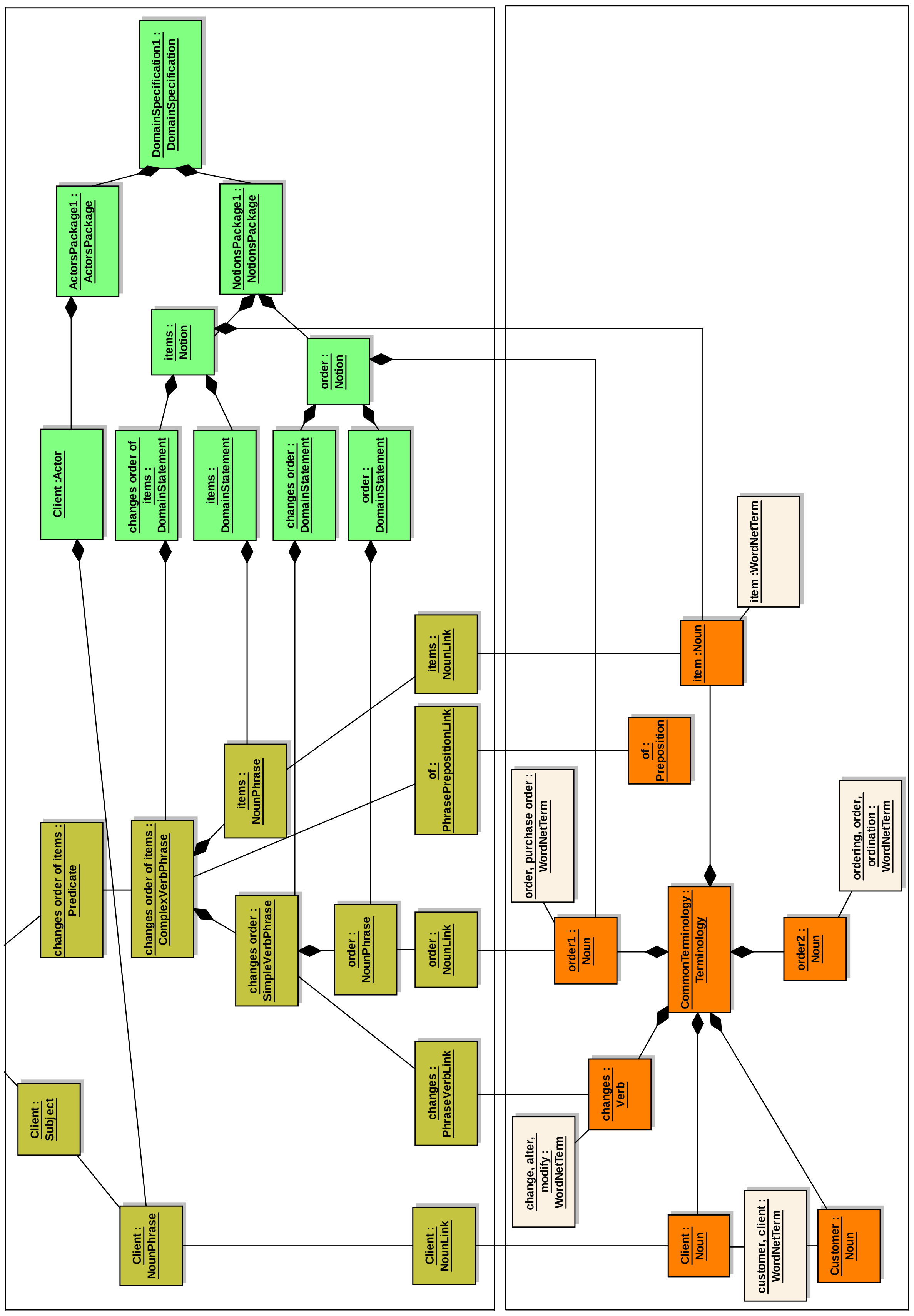
DomainSpecification1 : DomainSpecification
ActorsPackage1 : ActorsPackage
NotionsPackage1 : NotionsPackage
items : Notion
order : Notion
Client :Actor
changes order of items : DomainStatement
items : DomainStatement
changes order : DomainStatement
order : DomainStatement
changes order of items : Predicate
changes order of items : ComplexVerbPhrase
items : NounPhrase
changes order : SimpleVerbPhrase
order : NounPhrase
items : NounLink
of : PhrasePrepositionLink
order : NounLink
changes : PhraseVerbLink
Client : Subject
Client : NounPhrase
Client : NounLink
item :WordNetTerm
item :Noun
of : Preposition
order, purchase order : WordNetTerm
order1 : Noun
CommonTerminology : Terminology
order2 : Noun
ordering, order, ordination : WordNetTerm
changes : Verb
change, alter, modify : WordNetTerm
Client : Noun
customer, client : WordNetTerm
Customer : Noun

Figure 4.9: This figure shows the DomainSpecification of Figure 4.8 and the related elements of theTerminology.

Chapter 5

Similarity Measure Requirements

This chapter summarises the requirements that shall be met by the similarity measure. The chapter is structured in five sections. Firstly, requirements related to the query are explained (see Section 5.1). Section 5.2 illustrates which RSL entities should be compared to which other RSL entities. This aspect is defined in more detail in Section 5.5, while Section 5.3 specifies which results the similarity measure shall provide. Constraints on the requirements measurement are described in Section 5.4. All requirements are consequently numbered in order to allow for explicit references.

5.1 Queries

1. The similarity measure shall allow queries that specify:
   (a) a set of Sentences (Sentences may have different type) with associated DomainSpecification,
   (b) a set of Requirements with associated DomainSpecification,
   (c) a set of RequirementsPackages with associated DomainSpecification,
   (d) a complete RSL requirements model (consisting of Requirements and DomainSpecification),
   (e) a set of DomainElements (e.g. Notions, Actors),
   (f) a set of DomainElementPackages,
   (g) a complete DomainSpecification,
   (h) a set of Terms,

(i) any of the above entities combined with another one using the operators AND, OR, NOT.

2. The similarity measure shall be able to handle queries that contain different RequirementRepresentations mixed in a query.
3. The similarity measure shall allow us to specify the preferred size of cases contained in the results.

5.2 RSL Entities to be compared

4. Different sentence types: The similarity measure shall be able to calculate meaningful similarities for queries and cases that consist of different types of Sentences (e.g. NaturalLanguageSentence, SVOSentence, ModalSVOSentence, ConditionalSentence). The RSL entities that are used to represent a NaturalLanguageHypertextSentence on the one hand and a SVOSentence on the other hand are different. However, the meaning of such sentences may nevertheless be very similar.
5. Using RSL, a Sentence is more than simply a string. A Sentence consists of Phrases that are defined in the DomainSpecification. Phrases are linked to Terms in the Terminology and WordNetTerms. Figure 4.6 and Figure 4.9 exemplify the structure that might be related to a Sentence. Comparing two Sentences the information contained in this structure shall be considered in addition to the sentence string.
6. Different RequirementRepresentations: The similarity measure shall be able to calculate meaningful similarities for queries and cases that consist of a set of Requirements defined in different RequirementRepresentations. E.g. the query specifies a SentenceList with some NaturalLanguageHypertextSentences while one of the cases consists of several ConstrainedLanguageScenarios specifying the same requirement.
7. Size of Requirements: The similarity measure shall be able to compare one Requirement stated in the query with a set of Requirements contained in a software case and vice versa. A pairwise comparison of Requirements is not always sufficient in order to determine all existing similarities because of the different size of Requirements. Example: What is modelled in one ConstrainedLanguageScenario in Software Case A can be split into several Scenarios in Software Case B.
8. No configuration similarity: The similarity measure does not need to build configurations of software cases by itself.

9. Automatic fallback: Software cases with less formalized Requirements should be found as well, i.e., if a case has only NaturalLanguageRequirements and the query is specified in SVOSentences then a meaningful similarity has to be calculated for the case with the NaturalLanguageRequirements, too.

5.3 Results of the Similarity Measure

10. The results provided by the similarity measure shall allow us to list the cases according to their similarity with the query.
11. The result of the similarity measure shall illustrate commonalities and differences between the query and a certain case, e.g. which parts of the query and the case are identical, which are similar and which are completely different. E.g. the query consists of two Sentences and a case consists of 25 Sentences. Which of these 25 Sentences map to the query Sentences? A further example is illustrated in Figure 5.3. This is related to the solution marking mechanism, which will be developed in Task 4.3 (Definition of a solution marking mechanism for showing model differences). In contrast to the mechanism developed in Task 4.3 this requirement is only related to differences in requirements models. However, the same mechanisms can be used to achieve the required effect.
12. Results of the similarity measure:
    (a) $sim(query, case) \in [0, 1]$ where 0 denotes no similarity and 1 denotes an exact match.
    (b) If a case consists of all **or more** Requirements defined in the query and the Requirements are defined in the same RequirementRepresentation the result shall be 1. This is based on the assumption that the additional Requirements may possibly be reused and are therefore valuable for the Requirements Engineer. This requirement means that the similarity measure / function is not *commutative*. Assume that the query defines ten requirements and the case contains exactly the same requirements plus four others. Then $sim(query, case) = 1$ while $sim(case, query) < 1$.
    (c) If a case consists only of some of the Requirements specified in the query, the result shall be less than 1.
    (d) If a case consists of all Requirements defined in the query but the Requirements are defined in a different RequirementRepresentation the result shall be less than 1. (This is based on the assumption that it is more efficient to reuse Requirements already defined in the RequirementRepresentation used in the current project or for the aspect considered in the query.)

(e) If a case consists of all Requirements defined in the query but the DomainElements are mapped to different Terms or WordNetTerms the similarity shall be less than 1. -> Ambiguity problem, see Figure 5.1

(f) If a query and a case describe the same Requirement but in different words the similarity shall be high even if they do not have even one word in common. -> Paraphrase problem, see Figure 5.2

(g) If a query and a case each consists of a Requirement and the type of both Requirements is different, the similarity shall be less than 1.

(h) If a query and a case consist of a ConstrainedLanguageScenario with identical Sentences but the order of the Sentences is different then the similarity shall be less than 1.

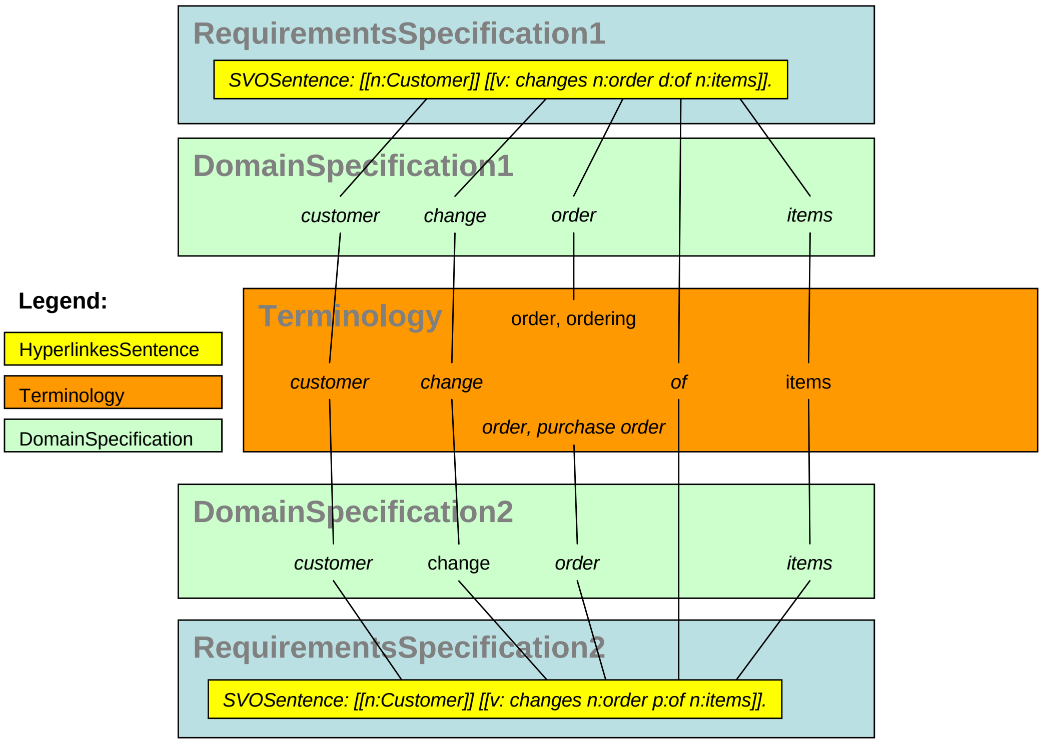

Figure 5.1: Ambiguity of Sentences: The sentences have the same wording but different meaning.

13. The results provided by the similarity measure shall allow us to illustrate which parts of a case are similar to the query and which not.

14. If the similarity is evaluated to be less than 1, the result of the similarity measure shall explain the cause, e.g. different Requirements, different RequirementRepresentations, different meaning of DomainElements, ...

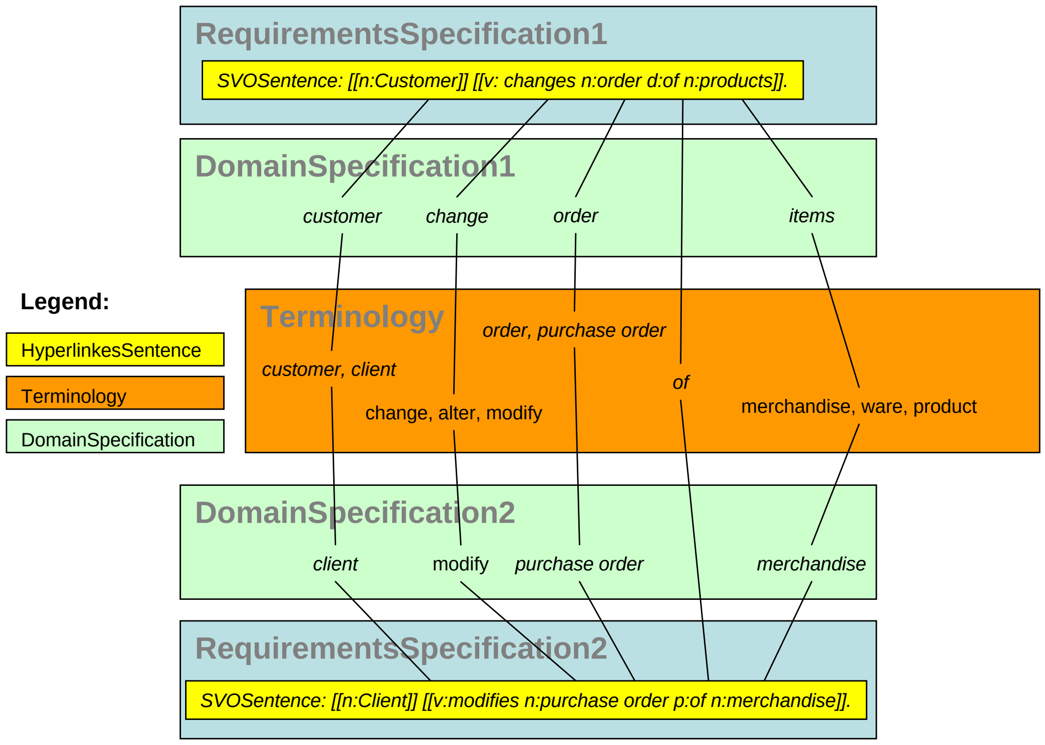

Figure 5.2: Paraphrase Problem: two sentences with similar meaning but different wording.

5.4 Constraints on Similarity Measure

15. The similarity measure shall require as little additional modelling effort as possible. Some similarity approaches evaluate the artefacts to be compared (e.g. requirements models, software cases) directly. Other approaches are based on characterisations of these artefacts. The creation of these characterisations can necessitate additional modelling effort if the information cannot be extracted automatically from the artefacts.

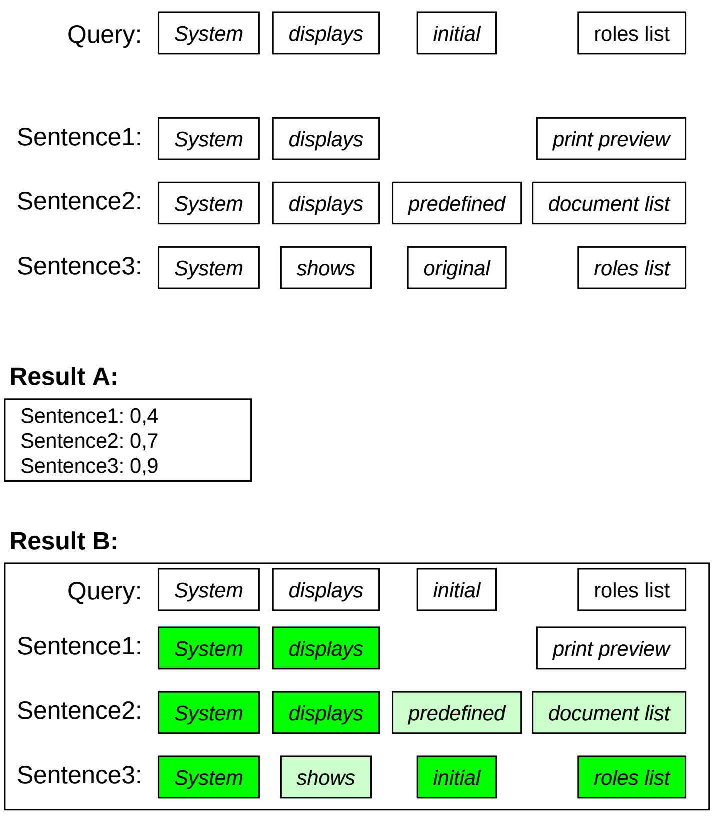
Query: System displays initial roles list
Sentence1: System displays print preview
Sentence2: System displays predefined document list
Sentence3: System shows original roles list
Result A:
Sentence1: 0,4
Sentence2: 0,7
Sentence3: 0,9
Result B:
Query: System displays initial roles list
Sentence1: System displays print preview
Sentence2: System displays predefined document list
Sentence3: System shows initial roles list

Figure 5.3: A similarity value for each case (see result A) is not sufficient to support the reuse of cases. Commonalities and differences should also be visualised (see result B for an example).

5.5 Detailed Requirements - RSL entities to be compared

An appropriate similarity measure can only be developed when it is known exactly what is to be compared. Preliminary statements regarding this topic were made in Section 5.2. In this section we define in detail which RSL entities have to be compared to which other RSL entities. The similarity measure need not be able to compare each RSL entity with each other RSL entity for two reasons. Firstly, not all RSL entities can be part of a query. For example all abstract classes of the RSL metamodel cannot be part of a query. Secondly, most RSL entities which may be part of a query only need to be compared to a few other RSL entities in order to determine similar cases.

We explain below, for the main RSL query entities, to which parts of RSL requirements models in the fact repository they should be compared. These results are summarised in several tables (see Figure 5.4 for an example). Each row shows, for one RSL query entity, to which RSL entities it should be compared. This is indicated by "m1, m2, m3, ..." in the table. The numbering makes it easy to refer to specific entries. For each entry an appropriate comparison method will be determined in the next chapter.

5.5.1 Requirements Specification

Each RSL query entity should be compared with the corresponding RSL entities in the software cases, i.e. if e.g. an Actor is specified in the query it should be compared with all Actor instances in the software cases. In the following we explain for each RSL entity to which other RSL entities it should be compared additionally. See Figure 5.4 for a summary of these explanations.

Complete RSL requirements model: A complete requirements model consists of a RequirementsSpecification and a DomainSpecification. The RSL entities to be compared with those specifications are specified below.

RequirementsSpecification: RequirementsSpecifications should be compared with RequirementsSpecifications.

RequirementsPackage: Due to the different size of cases the RequirementsPackage should be compared to RequirementsSpecifications also.

Requirement: A Requirement specified in the query should also be compared with RequirementsPackages since Requirements can vary in size. E.g. the content of one use case can be divided in several use cases connected with "invoke" in another RequirementsSpecification. This example holds also for other RequirementRepresentations. A SentenceList e.g. can be divided into several SentenceLists, too.

Query\Case	RequirementsSpecification	RequirementsPackage	Requirement
RequirementsSpecification	m1		
RequirementsPackage	m2	m3	
Requirement		m4	m5

Figure 5.4: The first column lists RSL entities that might be specified in a query. Each row shows to which RSL entities of software cases the specific RSL entity should be compared.

5.5.2 Requirement Representation

The same Requirement can be represented by different RequirementRepresentations in different software cases (see Requirement 6). Therefore, it is necessary to compare each RequirementRepresentation to each other RequirementRepresentation (see Figure 5.5).

Query \Case	SentenceList	ConstraintLanguageScenario	ActivityScenario	InteractionScenario
SentenceList	m1	m2	m3	m4
ConstraintLanguageScenario	m5	m6	m7	m8
ActivityScenario	m9	m10	m11	m12
InteractionScenario	m13	m14	m15	m16

Figure 5.5: All RequirementRepresentations need to be compared with all RequirementRepresentations.

Textual information plays a major role in all RequirementRepresentations, including the model-based RequirementRepresentations. RSL distinguishes nine types of Sentences. Figure 5.6 shows which RequirementRepresentations may contain which Sentence types.

SentenceList
NaturalLanguageHypertextSentence
SVOSentence
ModalSVOSentence
ConditionalSentence

ConstraintLanguageScenario
SVOSentence
InvocationSentence
PreconditionSentence
PostconditionSentence
ConditionSentence
RejoinSentence

ActivityScenario
SVOSentence
InvocationSentence
PreconditionSentence
PostconditionSentence
ConditionSentence
RejoinSentence

InteractionScenario
NounPhrase (part of NounPhraseLifeline)
VerbPhrase (part of PredicateMessage)
InvocationSentence
PreconditionSentence
PostconditionSentence
ConditionSentence
RejoinSentence

Figure 5.6: RequirementRepresentations and assigned the types of sentences they may contain.

It is possible to transform partially instances of ConstrainedLanguageScenario, ActivityScenario and InteractionScenario into each other. ConstrainedLanguageScenarios and ActivityScenarios consist of the same sentences' types. InteractionScenarios do not contain SVOSentences. However, InteractionScenarios do contain a NounPhrase and a VerbPhrase which are the major parts of SVOSentences. Thus, NounPhrase and VerbPhrase are listed in Figure 5.6 instead of SVOSentence.

Although the textual parts of ConstrainedLanguageScenarios, ActivityScenarios and InteractionScenarios can be mapped onto one another, it is not possible to define a 1-to-1 mapping between these RequirementRepresentations. Some information can only be specified in a particular RequirementRepresentation. For example, only the ConstrainedLanguageScenario contains a UIStoryboard.

NaturalLanguageHypertextSentence, ModalSVOSentence and ConditionalSentence may only be part of SentenceLists (see Figure 5.6). The only sentence type that may be part of all RequirementRepresentations is the SVOSentence. In the following we explain for each sentence type with which other RSL entities it should be compared.

5.5.3 Sentences

If a query contains a sentence then this sentence shall be compared with all sentences of the same type in any of the software cases (this is reflected by the markings in the diagonal in Figure 5.7). As stated in Requirement 4, different sentence types can be used to specify the same aspect of a Requirement. Thus, it is necessary to compare different types of sentences.

Different groups of sentences can be distinguished. ConditionalSentence, PreconditionSentence, PostconditionSentence and ConditionSentence all specify some kind of restriction. Thus, they should be compared with each other. This is indicated in Figure 5.7 by the m*i* with light grey background. While PreconditionSentence, PostconditionSentence and ConditionSentence, InvocationSentence and RejoinSentence define possible sequences of actions in UseCases the main sentence types are SVOSentence, ModalSVOSentence, ConditionalSentence and NaturalLanguageHypertextSentence. These main sentence types should be compared with each other as indicated in Figure 5.7 by the m*i* with dark grey background. The table in Figure 5.7 contains a NounPhrase and a VerbPhrase (parts of an InteractionScenario) since both together correspond to the SVOSentence contained in ConstrainedLanguageScenarios and ActivityScenarios.

A more uniform solution could be to compare all types of sentences with all other types of sentences. This approach has the disadvantage of a higher number of necessary comparisons, however. Whether it results in better similarity results can only be determined by experiments with a reasonable number of software cases.

Query \ Case	NaturalLanguageHypertextSentence	SVOSentence	ModalSVOSentence	ConditionalSentence	PreconditionSentence	PostconditionSentence	ConditionSentence	InvocationSentence	RejoinSentence	NounPhrase (part of NounPhraseLifeline)	VerbPhrase (part of PredicateMessage)
NaturalLanguageHypertextSentence	m1	m2	m3	m4						m5	m6
SVOSentence	m7	m8	m9	m10						m11	m12
ModalSVOSentence	m13	m14	m15	m16						m17	m18
ConditionalSentence	m19	m20	m21	m22	m23	m24	m25			m26	m27
PreconditionSentence				m28	m29	m30	m31				
PostconditionSentence				m32	m33	m34	m35				
ConditionSentence				m33	m34	m35	m36				
InvocationSentence								m37			
RejoinSentence									m38		
NounPhrase (part of NounPhraseLifeline)	m39	m40	m41	m42						m43	
VerbPhrase (part of PredicateMessage)	m44	m45	m46	m47							m48

Figure 5.7: The table shows to which types of sentences a specific query sentence should be compared.

5.5.4 Domain Specification

Besides the RequirementSpecification, the DomainSpecification is an important part of an RSL requirements model. Figure 5.8 lists the most important elements of DomainSpecifications. Like all other RSL entities DomainElements contained in a query should be compared with all instances of the same type contained in one of the software cases (see markings in the diagonal in Figure 5.8). Additionally the following RSL entities should be compared:

SystemElement and InputOutputDevice can be compared with each other because an object that is described as a SystemElement in a query may be specified as an InputOutputDevice in a software case and vice versa (comp. m12, m40 in Figure 5.8).

SimpleVerbPhrase should be compared with NounPhrases because even if another case does not comprise the SimpleVerbPhrase it may contain the NounPhrase contained in it e.g. the query specifies `store user data` and a certain software case only contains the NounPhrase `user data` (comp. m27, m29 in Figure 5.8).

ComplexVerbPhrase should be compared with SimpleVerbPhrase and NounPhrase because a case may contain these parts of a ComplexVerbPhrase even if it does not contain the whole ComplexVerbPhrase (comp. m32, m33 in Figure 5.8).

The words used in NaturalLanguageHypertextSentences are not necessarily mapped to Phrases in the DomainSpecification. In order to identify such occurrences, the main Elements of the DomainSpecification should be compared with all instances of NaturalLanguageHypertextSentences (see Figure 5.8). Two RSL entities may contain NaturalLanguageHypertextSentences, the SentenceList and the DomainElementRepresentation.

Query / Case	**DomainSpecification**	ActorsPackage	Actor	SystemElementsPackage	SystemElement	NotionsPackage	Notion	DomainStatement	NounPhrase	SimpleVerbPhrase	ComplexVerbPhrase	UIElement	InputOutputDevice	InputOutputType	UIContainer	**SentenceList**	NaturalLanguageHypertextSentence	**DomainElementRepresentation**	NaturalLanguageHypertextSentence
DomainSpecification	m1																		
ActorsPackage		m2															m3		m4
Actor			m5														m6		m7
SystemElementsPackage				m8													m9		m10
SystemElement					m11								m12				m13		m14
NotionsPackage						m15											m16		m17
Notion							m18										m19		m20
DomainStatement								m21									m22		m23
NounPhrase									m24								m25		m26
SimpleVerbPhrase									m27	m28	m29						m30		m31
ComplexVerbPhrase									m32	m33	m34						m35		m36
UIElement												m37					m38		m39
InputOutputDevice					m40								m41				m42		m43
InputOutputType														m44			m45		m46
UIContainer															m47		m48		m49

Figure 5.8: The table shows to which RSL entities a DomainElement contained in a query should be compared.

5.5.5 Terminology

A query can also specify Terms from the Terminology. Since Nouns, Verbs, Adjectives and Adverbs (both Modifier in Terminology) are the most meaningful entities of Sentences they should be compared with related parts in DomainSpecifications. See Figure 5.9 for a summary.

Nouns should be compared with all entities that can contain a Noun: Actor, SystemElement, Notion, DomainStatement, NounPhrase, SimpleVerbPhrase, ComplexVerbPhrase, UIElement, InputOutputDevice, InputOutputType and UIContainer (comp. m1-m11 in Figure

5.9). Additionally, Nouns should be compared with NaturalLanguageHypertextSentences since the words used in this type of Sentence might not be mapped to elements in the DomainSpecification.

Verbs should be compared with all entities that can contain a Verb: DomainStatement, SimpleVerbPhrase and ComplexVerbPhrase (comp. m14-m16 in Figure 5.9). Verbs should be compared with NaturalLanguageHypertextSentences for the same reason as stated for Nouns.

Modifier should be compared with NounPhrases since they may contain a Modifier. Additionally, the Modifier should be compared with NaturalLanguageHypertextSentences for the same reason as stated for Nouns.

Query / Case	**DomainSpecification**	ActorsPackage	Actor	SystemElementsPackage	SystemElement	NotionsPackage	Notion	DomainStatement	NounPhrase	SimpleVerbPhrase	ComplexVerbPhrase	UIElement	InputOutputDevice	InputOutputType	UIContainer	**SentenceList**	NaturalLanguageHypertextSentence	**DomainElementRepresentation**	NaturalLanguageHypertextSentence
Terminology																			
Noun			m1		m2		m3	m4	m5	m6	m7	m8	m9	m10	m11		m12		m13
Verb								m14		m15	m16						m17		m18
ModalVerb																			
Determiner																			
Modifier								m19	m20	m21	m22						m23		m24
Preposition																			
ConditionalConjunction																			

Figure 5.9: The table shows to which RSL entities a Term specified in a query shall be compared.

Chapter 6

Similarity Measures

This chapter describes similarity measures appropriate for comparing RSL requirements models. Different similarity measures are combined in order to conform with the different aspects of RSL requirement models and the requirements specified in Chapter 5.

Section 6.1 summarises the main characteristics of the similarity measures which are considered(see Chapter 3 for details). The Sections 6.2-6.5 describe the proposed similarity measures in more detail. Finally, in Section 6.6 these similarity measures are assigned to the RSL entities that need to be compared.

6.1 Comparison of similarity measures

The main characteristics of the similarity measures considered are summarised in Figure 6.1. Information Retrieval can be used as a fallback method because it can be applied to any artefact containing a significant amount of text. The method only provides lexicographic term matching and structure of artefacts is not evaluated. Thus, a more elaborate method should be applied when possible. Structural Case-based Reasoning applies a fixed structure for comparing cases while ReDSeeDS software cases have a variable structure. The RSL metamodel defines all possible structures for RSL requirements models. Graph-based comparison and Description Logic can be used to compare the structure of elements. They process elements of different structure. For comparing pure text, neither approach might be the best choice, however. NaturalLanguageHypertextSentences might consist of pure text, if the contained words are not mapped to elements of the DomainSpecification. The graph-based similarity approach directly evaluates the software cases stored in the fact repository while all other approaches need a preprocessing of the data stored in the fact repository.

This brief comparison illustrates that there is no approach that is best suited to all aspects of RSL requirements models. A combination of these approaches is necessary to cope with all parts of these models. The following sections explain how the similarity approaches introduced in Chapter 3 can be adapted to the needs of ReDSeeDS.

Approach	**Information Retrieval**	**Structural CBR**	**Description Logic**	**Graph-based Similarity**
Elements compared	▪ text-based documents ▪ structure is not analysed by state-of-the-art approaches	▪ cases of identical structure (attributes) with different values for the attributes	▪ concept definitions that are consistent with RSL metamodel (contained in TBox)	▪ graphs of any structure
Input required	▪ document representations needed ▪ automated generation of representations possible (Stemming, Indexing, etc.)	▪ case representations needed ▪ automated generation of representations possible ▪ additional information can be added	▪ OWL representation of queries and cases needed ▪ JGraLab -> OWL converter has been developed by University of Koblenz	▪ case ▪ no generation of representations necessary
Result provided	▪ Value in [0..1]	▪ Value in [0..1]	▪ Value in [0..1]	▪ Value in [0..1] ▪ One integrated graph containing both query and case
Underlying meaning of "Similarity"	Documents containing same strings in the same frequency are very similar / identical.	Cases with same values for attributes are very similar / identical.	Concept definitions containing concepts with minimal distance in the taxonomy and the same roles are very similar / identical.	Cases consisting of the same graph elements (local similarity determined by attributes) arranged in the same structure (interconnections) are very similar / identical.
Paraphrase Problem	not solved by state-of-the-art approaches; topic of current research	can be solved by local similarity measure that evaluates the mapping to Terms in Terminology	can be solved by integrating the Terminology in the Tbox	can be solved by integrating the Terminology in the fact repository

Figure 6.1: Comparison of Similarity measures with respect to their application in ReDSeeDS.

6.2 Global Similarity Measures

The global similarity combines the different local similarities. For this purpose, the local similarities are grouped by the major elements of a software case, i.e., requirements, architecture, detailed design, and code. Thus, we use the standard weighted-normalized-sum similarity measure (see Section 3.3) at two levels.

For the purpose of the global similarity, we distinguish between "items" as non-technical properties of a software case and "attributes" as technical properties in the characterization[1] for the purpose of similarity calculations. "Technical properties" refers to properties that are used by the similarity engine for calculating similarities. Their representation might be optimized for similarity calculation and, therefore, need not be in any of the ReDSeeDS languages.

For a software case, the following items are subject to a characterization:[2]

- Requirements items: (following the different requirement types from Section 4.1)
  - FunctionalRequirementsOnSystem
  - FunctionalRequirementsOnComposite
  - ConstraintOnSystem
  - UseCase
  - DomainSpecification
- Architecture items:
  - Architectural style
- DetailedDesign items:
  - Design patterns used
- Code items:
  - Programming language

[1] see ReDSeeDS Deliverable D4.1 (Section 4.2) for a definition and explanation of "characterization" and "artifact"

[2] This structure might change with actual cases becoming available for experimentation and is, therefore, a "first guess".

The global similarity is then the weighted normalized sum of the local similarities for these items, grouped by requirements, architecture, design, and code:

$$\begin{aligned} sim(Q,C) \; = \; & (w_r \frac{\sum_{Q.r.item \neq undef, C.r.item \neq undef} w_{r.item} \cdot sim_{r.item}(Q.r.item, C.r.item)}{\sum_{Q.r.item \neq undef, C.r.item \neq undef} w_{r.item}} \\ & + w_a \frac{\sum_{Q.a.item \neq undef, C.a.item \neq undef} w_{a.item} \cdot sim_{a.item}(Q.a.item, C.a.item)}{\sum_{Q.a.item \neq undef, C.a.item \neq undef} w_{a.item}} \\ & + w_d \frac{\sum_{Q.d.item \neq undef, C.d.item \neq undef} w_{d.item} \cdot sim_{d.item}(Q.d.item, C.d.item)}{\sum_{Q.d.item \neq undef, C.d.item \neq undef} w_{d.item}} \\ & + w_c \frac{\sum_{Q.c.item \neq undef, C.c.item \neq undef} w_{c.item} \cdot sim_{c.item}(Q.c.item, C.c.item)}{\sum_{Q.c.item \neq undef, C.c.item \neq undef} w_{c.item}}) \\ & /(w_r + w_a + w_d + w_c) \end{aligned} \tag{6.1}$$

Explanations:

- Q: the query case as software case
- C: the reuse candidate as software case
- r: requirements
- a: architecture
- d: detailed design
- c: code
- e: one of r, a, d, c
- w_r, w_a, w_d, w_c: weights for the different major elements, i.e., requirements, architecture, design, code.
- $w_{e.item}$: weights for the different items in a major element
- $Q.r.item$ refers to an item "item" in the requirements section of a query
- $C.a.item$ refers to an item "item" in the architecture section of a software case
- $sim_{r.item}$ refers to the local similarity measures for a certain item - here the item "item" in the requirements. A local similarity measure always delivers a value between 0 and 1 (see 5.3).

As similarity measures for the items under architecture, design, and code, some rather standard similarity measures from CBR could be employed: Taxonomic similarity measures for patterns, i.e. items "design patterns" and "architectural style": symbol similarity measures for "programming language". Thus, for our research, we focus on the new similarities for requirements. These also relate most to the subject of the project, i.e. requirements-oriented reuse.

For *overlapping similarity measures* for an item, we use the weighted normalized sum to weight different local similarities. The actual characterization used for similarity computation might contain different attributes for the different local similarities for an item.

$$sim_{e.item}(Q.e.item, C.e.item) = \frac{\sum_{i=1..n} w_i \cdot sim_{e.item,i}(Q.e.item, C.e.item)}{\sum_{i=1..n} w_i} \qquad (6.2)$$

Knowledge of which attribute belongs to which item is encoded in the similarity functions. The local similarity for overlapping similarity measures has the following properties:

- Information / data is available for all similarity measures: The similarity value from each measure takes a part of the weight of the item.
- Information / data is not available for all similarity measures: A fallback mechanism (comp. Requirement 4 and 9 in Section 5.2) is implemented by the above formula together with a characterization extractor that maps more formal representations on less formal representations (e.g., ConstraintLanguageScenarios on simple text). To satisfy requirement 12d, the formula also counts undefined attributes - in contrast to the global similarity for items (formula 6.1).

6.3 Textual Similarity Measures

The idea of the textual similarity measures is to compare Requirements or Scenarios based on the text that they contain or represent. Thus, respective characterization extraction has to extract a list of words from the given Requirement(s) or Scenario(s). The advantage of the measure is that it can be applied to almost any RequirementsSpecification. This is obvious for ConstraintLanguageSentences. ActivityScenarios also consist of SVOSentence and, therefore, also can be mapped to text. Respectively, InteractionScenarios can be mapped to text. Thus, all types of RequirementsSpecifications can be mapped to text and therefore compared using a textual similarity measure.

In the following, we present two textual measures that can be used for measuring the similarity of requirements and/or scenarios: The set-of-words measure (6.3.1) does not consider the structure at all. The set-of-sentences measure (6.3.3) considers the structure introduced by sentences.

6.3.1 Similarity using Set-of-Words Measure

The set-of-words measure weights the occurrences of words equally. No structure is considered for the similarity calculation.

The actual similarity measure is composed of two parts: A similarity at the word list level combines similarities at the word level.

For the *similarity at the word list level*, we use a set similarity measure. Thus, the order of words is not considered. Furthermore, we use a set similarity measure that does not consider word frequencies and document length in order to not conflict with requirement 12b. A set similarity measure with such properties has been developed by Tautz & Gresse van Wangenheim [TG98]:

$$sim_{setOfWords}(q,c) = \begin{cases} \sum_{e_1 \in q} \frac{max\{sim_{t_{attr}}(e_1,e_2)|e_2 \in c\}}{card(q)} & \Leftrightarrow card(q) > 0 \wedge card(c) > 0 \\ 0 & \Leftrightarrow card(c) = 0 \\ 1 & otherwise \end{cases} \tag{6.3}$$

For the *word-level similarities*, the following similarities and mechanisms can be used and combined:

Boolean similarity measure: For comparing two words, the boolean similarity reflects the result of a string comparison:

$$sim(e_q,e_c) = \begin{cases} 1 & \Leftrightarrow e_q = e_c \\ 0 & otherwise \end{cases}$$

Stopword removal mechanisms: Stopwords are frequent words that have no specific meaning in a domain (e.g., a, the, of). Since such words usually occur in all documents, they do not help to determine the relevance of a document and could even mislead the similarity when weighted like any other word in a query. Thus, stopwords should be removed.

Stemming mechanisms: In linguistics, a stem is the part of a word that is common to all its inflected variants. Stemming is the process of reducing inflected (or sometimes derived) words to their stem, base or root form – generally a written word form. The words can then be compared based on their stem. Thus, stemming can improve the results of the above boolean similarity.
Obviously, stemming depends on the language: In the English language a simple stemming by cutting off suffixes works pretty well – e.g., cutting off the plural "s". There are several such stemming algorithms (e.g. Paice / Husk Stemmer[3]). An exception is irregular verbs. In the German language, such a simple stemming does not work very well because when the inflection changes, so may characters in the stem part of the word, e.g., "Baum" and "Bäume".

N-gram matching: N-Gram Matching compares words using N-character sub-words, e.g. 3-gram matching using 3-character sub-words. This allows to (1) compensate typos and (2) implement a language-independent stemming. Problems are the high complexity and high memory need. So, there are certain volume limits for n-gram matching.

Synonym-based similarity measure: Using a thesaurus, synonyms can be found. With a fixed synonym similarity (like the WordNet synonyms), the boolean similarity measure can be refined as follows:

$$sim(e_q, e_c) = \begin{cases} 1 & \Leftrightarrow e_q = e_c \\ sim_{synonym} & \Leftrightarrow isDirectSynonym(e_q, e_c) \\ 0 & otherwise \end{cases}$$

where

- $sim_{synonym}$: synonym similarity $\in]0;1[$ (constant)
- $isDirectSynonym(e_q, e_c)$: true if the word e_q is a direct synonym of the word e_c

The synonym-based similarity measure can address the paraphrase problem (requirement 12f) by exploiting the mappings between different domains as synonym specifications for this measure.

Taxonomy-based similarity measures A taxonomy adds relationship types[4] to the vocabulary. WordNet-based similarity measures (see Section 3.2) are an example of this type of measure. For taxonomies as trees with a uniform relationship type, heuristics using the the node depth are meaningful, e.g. from REFSENO [TG98]:

$$sim(e_q, e_c) = \frac{d(cnode(e_c, e_q))}{d(e_c) < d(e_q) ? d(e_c) : d(e_q)}$$

[3]http://www.comp.lancs.ac.uk/computing/research/stemming/Links/paice.htm
[4]Some terminology representations or thesauri also allow and provide other relationships than "is synonym".

where

- $d(n)$: depth of node n where $d(ROOT) = 1$
- $cnode(n1, n2)$: least common subsumer of the nodes $n1$ and $n2$ in the taxonomy

Bergmann has described how to set up taxonomy similarities for various relationship types [Ber98].
Another taxonomy similarity is described below as "distance-based similarity" (Section 6.4.1). For computational reasons, it may be necessary to maintain a query-word-by-case-word similarity table to allow a fast lookup of word similarities. However, such a table must be updated upon changes to the terminology.
The synonym-based similarity measure can address the paraphrase problem (requirement 12f) by exploiting the mappings between different domains as synonym specifications for this measure.

6.3.2 Known Issues

Utility modeled by similarity The smaller the query, the higher the risk that cases with a similarity of 1 are not relevant (typical information retrieval issue with short queries).

Performance Set measures posed performance problems in early CBR systems. It will be necessary to determine by experiments to which text sizes the set-of-words measure is applicable with reasonable performance.

6.3.3 Similarity using a Set-of-Sentences Measure

The idea of the set-of-sentences measure is to consider the requirements structure for search – in contrast to the set-of-words measure. Thus, we use a set similarity measure on top of the set-of-words measure:

$$sim_{setOfSentences}(q,c) = sim_{set}(q,c)\ with\ sim_{t_a ttr} := sim_{setOfWords}$$

The set-similarity measure [TG98] is fed with each sentence as an element and operates on this set of sentences. For each sentence, the set-of-words similarity is used. This allows for the variations in the word similarity described with the set-of-words similarity measure.

6.3.4 Known Issues

Impact of structure measure The actual impact of the structure measure is not obvious. The measure should improve the precision of comparison of SVO sentences because words are not mixed from different sentences as is the case for the set-of-words measure on whole scenarios. However, with natural language, a sentence could also contain several SVO statements. In general, for small collections (<100 cases), we do not expect a significant impact on the ranking order in comparison to the pure set-of-words measure on sentences.

Performance Set measures posed performance problems in early CBR systems. Due to the two-recursion set-similarity construction of this measure, it is expected be more sensitive to performance issues than the set-of-words measure. It will be necessary to analyze by experiments to which text sizes the set-of-words measure is still applicable.

6.4 Description Logic-based Similarity Measures

Measuring similarity in Description Logics (DLs) is based on comparing two concept definitions. This comparison includes both the concept placement in the taxonomic hierarchy and the roles and role fillers that the concepts define (see also Section 3.4).

6.4.1 Algorithms for Similarity Measures for Conceptual Structures

In this section we provide algorithms that can be applied in Description Logic systems for measuring similarity of two conceptual structures. All algorithms are denoted in pseudo code, i.e. implementation-independent.

Concept-based Similarity

Concept-based similarity describes the similarity of two concepts based on two aspects. The first aspect is their position in the taxonomy and the second aspect is their roles. Both aspects can be computed independently from each other and therefore two algorithms describing the similarity of distance between concepts and common roles of concepts are computed and the returned similarity values are summed up.

```
ALGORITHM: conceptBasedSimilarity(c1, c2)
```

```
1. return $\frac{\text{distanceBasedSimilarity(c1, c2) + roleBasedSimilarity(c1, c2)}}{2}$
```

Distance-based Similarity

Distance-based similarity describes the similarity of two concepts based on their position in the taxonomy. The distance of both concepts to the least common subsumer (LCS) is computed, both values are added and a value describing distance-based similarity is returned: 1 means both concepts are in fact the same concept and the smaller the value the further away the concepts are from each other.

ALGORITHM: distanceBasedSimilarity(c1, c2)

```
1. lcs = leastCommonSubsumer(c1, c2)
2. distance1 = pathLength(c1, lcs)
3. distance2 = pathLength(c2, lcs)
4. distance = distance1 + distance2
5. return $\frac{1}{\alpha+\text{distance}}$
```

Note that the α value is needed because comparing a concept to itself the taxonomic distance is 0 and division by 0 is not allowed. See also Section 6.4.2 for more details.

Role-based Similarity

Role-based similarity describes the similarity of two concepts based on their roles and role fillers. The roles of both concepts are recursively compared and similarities are summed up for every concept. A value describing role-based similarity is returned: 1 means both concepts have the same roles and the smaller the value the less their roles have in common.

The similarity is based on values that are computed by comparing every role of one concept to every role of the other concept. A matrix is filled with all similarity values and the maximum value in every row or column indicates the best matching roles of both concepts. In order to find a match for every role and not use the same role as a match more than once, when $c1$ has less roles the maximum value of a column is used and when $c2$ has less roles the maximum value of a row is used.

ALGORITHM: roleBasedSimilarity(c1, c2)

```
1. if c1 OR c2 has roles, do
   (a) similarity = 0
   (b) roleValues = [ |roles(c1)| ], [ |roles(c2)| ]
   (c) for int i=0; i < |roles(c1)|; i++, do
       i. s = roles(c1)[i]
       ii. for int j=0; j < |roles(c2)|; j++, do
           A. t = roles(c2)[j]
           B. roleValues[i][j] = conceptBasedSimilarity(s, t)
   (d) if |roles(c1)| < |roles(c2)|
       i. for int k=0; k <= |roles(c1)|; k++, do
           A. similarity += max(roleValues[k][*])
   (e) else
       i. for int k=0; k <= |roles(c2)|; k++, do
           A. similarity += max(roleValues[*][k])
   (f) similarity = $\frac{\text{similarity}}{\max(i,j)}$
2. else, do
   (a) similarity = distanceBasedSimilarity(c1, c2)
3. return similarity
```

6.4.2 Known Issues

There are some issues that have to be taken into account and may only be assessed and ruled out during development of the ReDSeeDS engine, i.e. after implementing and evaluating the algorithms.

- In the algorithm for distance-based similarity, some value α needs to be added to the distance due to the fact that when comparing a concept to itself the distance is 0 and division by 0 is not allowed. What exactly is a good value for α needs to be clarified during implementation and experiments. Another possibility to measure the distance of two concepts is to compute the product of the number of siblings for every predecessor on the path from the starting concept to the least common subsumer. For this distance measure the number of siblings is taken into account which is based on the assumption that two siblings are more similar when there are no further siblings (all instances of the parent belong to either of the two) and are less similar when there are further siblings (all instances of the parent belong to one of the siblings).

- An addition to the above mentioned algorithm of measuring role-based similarity is to consider not only the role filler but also the role name. When, for example, an SVOSentence is compared to another SVOSentence, the types of roles that have to be compared are known: the Noun of the one Sentence needs to be compared to the Noun of the other Sentence, and analogously the Verbs and Objects are compared. This means that taking the role name into account may improve the comparison of matching roles between the two concept structures. However, completely relying on the role name is dangerous: when comparing two concept structures that are of different types (e.g. an SVOSentence and a NaturalLanguageSentence) the role names are different, but nonetheless the concept structures may be semantically equivalent. Furthermore, if matching roles with different names refer to the exact same filler, it may be reasonable to treat them as equal or as more similar than two roles with the same name and different fillers. How much weight a role name should have for the comparison of concepts, and for which combinations of RSL entities, has to be evaluated during implementation with concrete examples.

- Roles can be used to denote simple attributes of concept, like a name, ID, etc. Such attributes are modeled with so-called concrete domains, i.e. strings, integer numbers, real numbers, sets of strings, integers or reals and ranges of integers or reals. Different mathematical approaches are needed to compare the different concrete domains. For example, simple integer or real numbers can be easily compared by relying on the obvious semantics of the binary mathematical relations =, $\neq$, <, $\leq$, >, and $\geq$. However, the distance between two numbers may be more significant than knowing that one number is greater than another one. For sets and ranges of numbers the intersection of the compared domains is interesting: two concrete domains are assumed to be more similar when they have more elements in common. However, the number of elements in a set and the distance of numbers are of interest, too, because $\{1,4,7\}$ is more similar to $\{2,5,8\}$ than to $\{0,1\}$, although they have no common elements.

- When comparing a role filler of a concept with all role fillers of another concept, it may happen that two or more combinations of fillers are assigned the same similarity value. Automatically selecting the "best matching" role filler is ambiguous and may be non-deterministic for such situations. It is not yet known if and how often such situations will be encountered, and how they may be resolved. Hence a detailed evaluation of this issue is postponed until a working implementation and concrete examples are present.

6.4.3 Example

Figure 6.2 simplifies the UML class diagram from Figure 4.8 to improve the overview. The structure of SVOSentences involving NounPhrases, Predicates, VerbPhrases is omitted. To be able to see of which words the two example sentences are constructed, the Links are kept. However, the most important aspect of this figure is the Terminology for this example containing Terms and WordNetTerms.

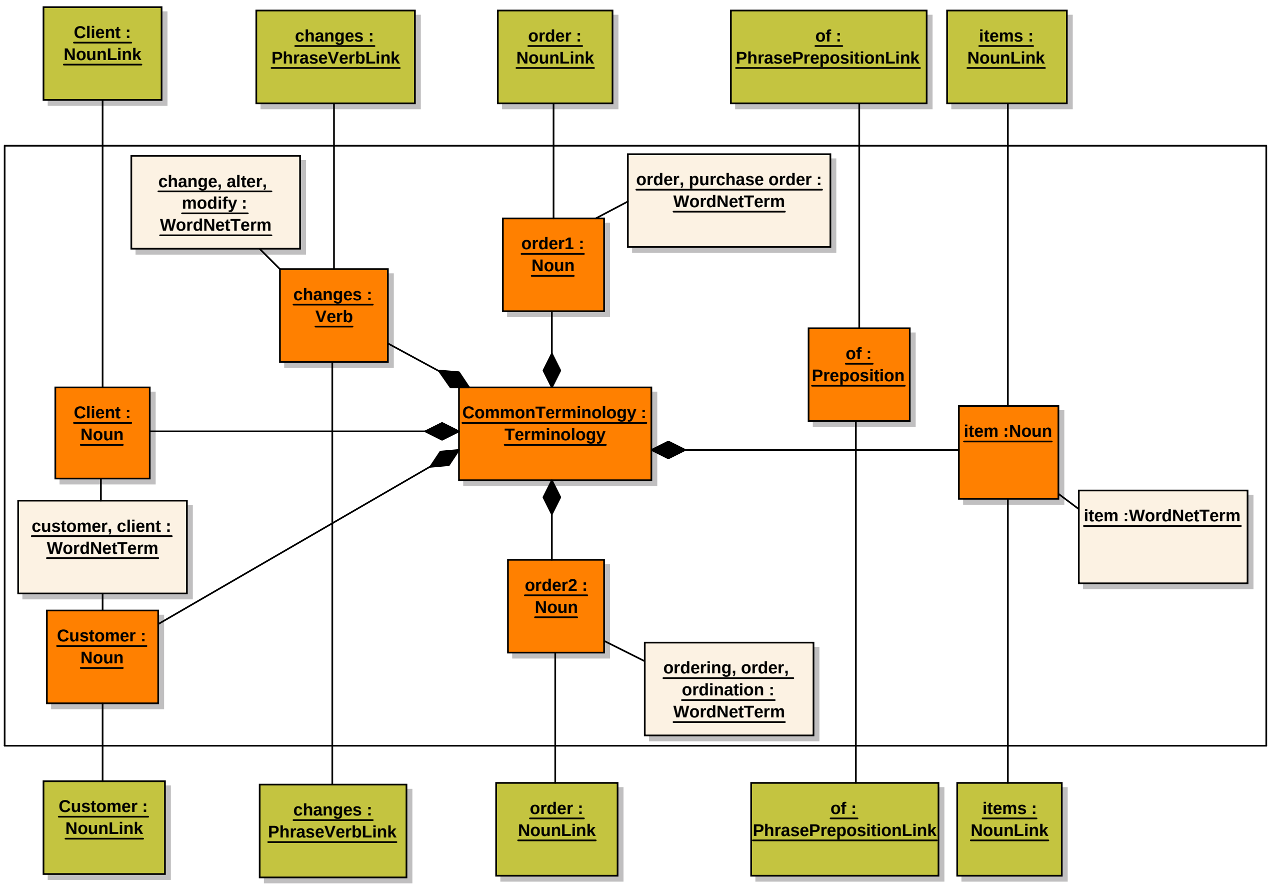

Figure 6.2: Two SVOSentences and their relations to WordNetTerms (excerpt from the example in Figure 4.9 on page 28).

A taxonomy of WordNetTerm instances was constructed that contains the corresponding terms of the example sentences. *WordNet* defines terms with a *hyponomy-of* relation that has been translated into a *is-a* relation, i.e. taxonomy between Terms (see Figure 6.3).

The query is formulated as an instance of a RequirementsSpecification, i.e. with instances of the corresponding UML classes of the RSL. Instance recognition is applied to compute the most specific concept of which every instance is an instance. Then these concepts are compared using the concept-based similarity measure. The cases are modeled as concepts in the TBox but, to stay as close as possible to the other approaches and previous figures, we use instances of the corresponding UML classes in Figures 6.2 and 6.3.

It is easy to see that the Sentences use both common and different Terms. The Sentences use four common Terms and one different Term. Note that `Client` and `Customer` are synonyms that link to the same WordNetTerm, while `order1` and `order2` are used differently in both sentences, as homonyms: they link to different WordNetTerms. Measuring the similarity of both Sentences should compute quite a high value, i.e. a value less than but still close to 1. Using the concept-based similarity measure, we compute their similarity step by step in the following.

First, we compute the distance-based similarities of Terms that are used in both Sentences. For reasons of simplicity we assume that the structure of the Sentences is relevant and known. Thus, Subjects are compared to Subjects, Verbs are compared to Verbs, and so on. Not every role of one Sentence has to be compared to every role of the other Sentence and thus the matrix of similarity measures that is described in the role-based similarity algorithm need not be completely filled. Instead, the best matches are known and $\max(\text{roleValues}[i][j])$ can be abbreviated with $\text{matchingRoleValue}[i][j]$. The α of the distance-based similarity algorithm uses the value 0 in this example.

- `Client` and `Customer` are both Nouns, hence are specialisations of Noun and thus siblings in the taxonomy: their taxonomic distance is 2 and distanceBasedSimilarity(`Client`, `Customer`) = $\frac{1}{2}$.
- Both Sentences use the Verb `change`, i.e. the same Term. There is no taxonomic distance when comparing a concept to itself and distanceBasedSimilarity(`change`, `change`) = 1.
- The Nouns `order1` and `order2` are used, each in one of the two Sentences. Since they are both specialisations of Noun, the taxonomic distance is 2 and distanceBasedSimilarity(`order1`, `order2`) = $\frac{1}{2}$.

- Both Sentences use the Preposition `of`, i.e. the same Term. There is no taxonomic distance when comparing a concept to itself and distanceBasedSimilarity(`of`, `of`) = 1.
- Both Sentences use the Noun `item`, i.e. the same Term. There is no taxonomic distance when comparing a concept to itself and distanceBasedSimilarity(`item`, `item`) = 1.

Next, we compute the role-based similarities of the Terms. Note that all Terms specify a role with the filler CommonTerminology; to be more exact this is an inverse role because the Terminology consists of the Terms. But since all Terms specify this role with the same filler we omit it during the following evaluation.

- `Client` and `Customer` both define a role that has the same filler: the WordNetTerm `customer, client`. This means that, although the two Nouns have a different name, roleBasedSimilarity(`Client`, `Customer`) = 1.
- Both Sentences use the Verb `change`, i.e. the same Term. It is apparent that roleBasedSimilarity(`change`, `change`) = 1.
- The Nouns `order1` and `order2` both have a role with a specific WordNetTerm as filler: the first Sentence has filler `order, purchase order`, the second Sentence has filler `ordering, order, ordination`. The least common superconcept of the two WordNetTerms is `abstraction, abstract entity` (see also Figure 6.3). The taxonomic distance between `order, purchase order` and `abstraction, abstract entity` is 6 and the taxonomic distance between `ordering, order, ordination` and `abstraction, abstract entity` is 3. Thus the distance between `order, purchase order` and `ordering, order, ordination` is 9 and roleBasedSimilarity(`order1`, `order2`) = $\frac{1}{9}$.
- Both Sentences use the Preposition `of`, i.e. the same Term. It is apparent that roleBasedSimilarity(`of`, `of`) = 1.
- Both Sentences use the Noun `item`, i.e. the same Term. It is apparent that roleBasedSimilarity(`item`, `item`) = 1.

The role-based similarity of two Sentences is the sum of concept-based similarities between their roles (i.e. distance-based similarity plus role-based similarity divided by 2 per matching role), divided by the number of roles:

$$\begin{aligned}\frac{\frac{\frac{1}{2}+1}{2}+\frac{1+1}{2}+\frac{\frac{1}{2}+\frac{1}{9}}{2}+\frac{1+1}{2}+\frac{1+1}{2}}{5} &\\ = \frac{\frac{3}{4}+1+\frac{11}{36}+1+1}{5} &\\ = \frac{4.0\bar{5}}{5} &\\ = 0.8\bar{1}\end{aligned}$$

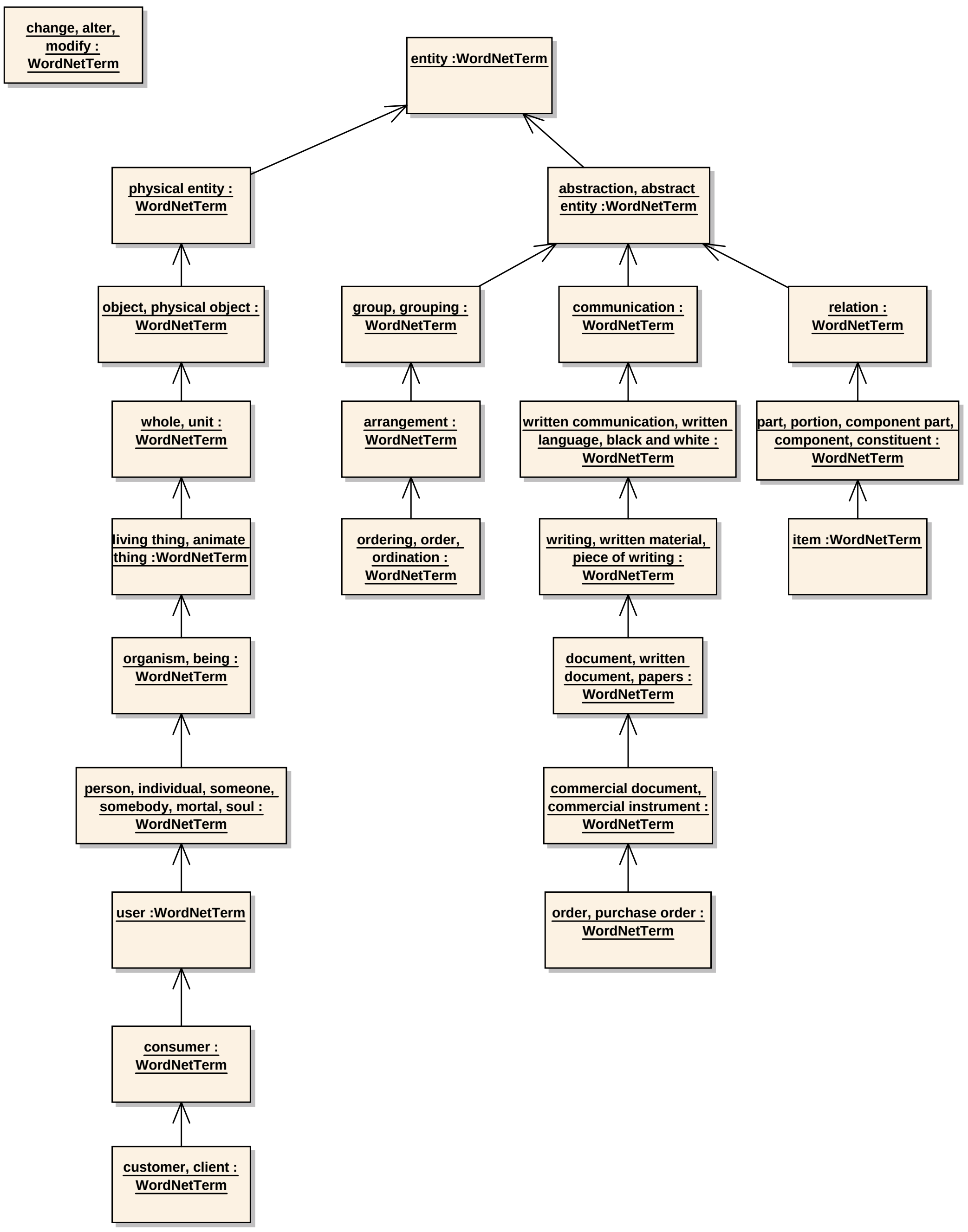
change, alter, modify : WordNetTerm
entity :WordNetTerm
physical entity : WordNetTerm
abstraction, abstract entity :WordNetTerm
object, physical object : WordNetTerm
group, grouping : WordNetTerm
communication : WordNetTerm
relation : WordNetTerm
whole, unit : WordNetTerm
arrangement : WordNetTerm
written communication, written language, black and white : WordNetTerm
part, portion, component part, component, constituent : WordNetTerm
living thing, animate thing :WordNetTerm
ordering, order, ordination : WordNetTerm
writing, written material, piece of writing : WordNetTerm
item :WordNetTerm
organism, being : WordNetTerm
document, written document, papers : WordNetTerm
person, individual, someone, somebody, mortal, soul : WordNetTerm
commercial document, commercial instrument : WordNetTerm
user :WordNetTerm
order, purchase order : WordNetTerm
consumer : WordNetTerm
customer, client : WordNetTerm

Figure 6.3: WordNet taxonomy of Nouns for Terms occurring in the example SVOSentences.

Finally, the concept-based similarity of the two SVOSentences would additionally include computing the distance-based similarity of the two Sentences and take the complete structure shown in Figure 4.8 into account. For reasons of simplicity we omit the complete structure for this example.

If, for example, the two Sentences used the same notion of `order` so that their only difference is the synonym of `Client` and `Customer`, the similarity would increase to $\frac{\frac{3}{4}+1+1+1+1}{5} = \frac{4.75}{5} = 0.95$. The value 1 is not reached because `Client` and `Customer` are still two different Nouns (represented by different UML classes), although being synonyms. If, instead of a synonym, two different Subjects were used, obviously the similarity value would decrease.

6.5 Graph-Based Similarity

The graph-based similarity of two elements A and B is based on comparison of the abstract syntax graphs of the two elements. As mentioned in Section 3.5, the currently most promising graph-based similarity measure is the similarity flooding algorithm [MGMR02] and its variants. Due to a close cooperation between the Universities of Koblenz and Siegen, the SiDiff algorithm [UK05] is designated for use in the ReDSeeDS engine. The remainder of this section will explain the basics of this algorithm and the metamodel for the graphs the algorithm works on.

6.5.1 SiDiff Metamodel

The SiDiff algorithm works on graphs that are instances of the SiDiff metamodel, which is depicted in figure 6.4. To be independent of existing and changing metamodels and to allow the SiDiff algorithm to compare models as instances of a wide range of metamodels, the SiDiff metamodel is very general.

A Document consists of several Elements. In a UML class diagram, a Document contains all Elements of the model. Each of the Elements has exactly one ElementType. Beside this, Elements might also contain several Attributes. Every Attribute models a name / value pair.

The major part of the graph structure is modeled by the reflexive composition at Element. This composition builds up a tree structure of the document. Elements may contain sub-Elements, for instance in UML a Package contains Classes, a Class contains Methods, a Method contains Parameters and so on. The second part of the graph structure is model by the class Reference.

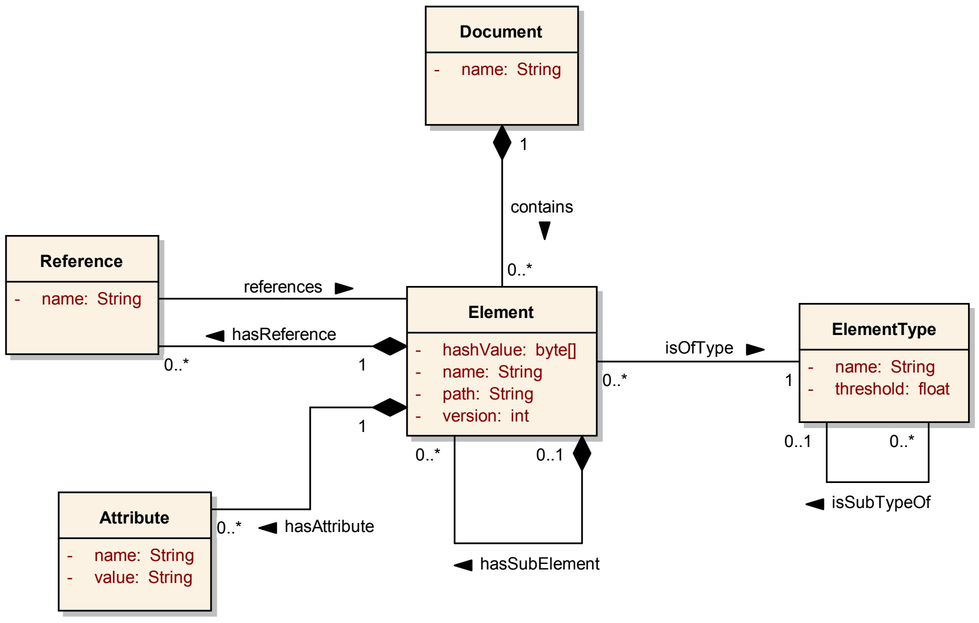

Figure 6.4: Metamodel used by SiDiff

References may exist between any two Elements. An example of a Reference in UML is an Association between two Classes that is not a Composition.

6.5.2 The SiDiff Algorithm

The algorithm used by the SiDiff tool to measure the difference of two models *A* and *B* can be divided into two phases, a bottom-up and a top-down one.

In the first one, similar model elements are identified in both models. This phase works bottom-up and only elements that are similar to exactly one other element in the second model are matched. To control the match, a similarity threshold can be specified for each element type in a model. Thus, only elements with a similarity greater than this threshold are matched in the bottom-up phase. If an element is similar to more than one element in the second model, the bottom-up phase does not match the elements. Instead, the matching is deferred to a later phase of the algorithm.

As soon as two elements are matched, the algorithm checks if child elements exist that have not been matched. If one such child element is found, the algorithm switches to the top-down phase. In that phase the last match of the bottom-up phase is propagated to all child elements of the matched elements. This leads to new similarity values for the child elements, since the match of the parent elements is taken into account. On the basis of these new similarity values, additional elements can be matched. These matches are further propagated top-down. After no more child nodes can be matched, the algorithm switches back to the bottom-up phase.

As soon as all elements have been tried to be matched in the bottom-up phase, the algorithm stops. The result of the algorithm is a table which contains the matching element pairs and their similarity. Using this table, a unified model containing all elements of both models can be created. Besides this, and more interesting for the similarity calculation of Software Cases, the resulting table can be used to calculate a real value that depicts the similarity of two models. The details of this calculation have already been depicted in section 3.5.

6.6 Applicability of Similarity Measures

In Section 5.5 we described in detail which RSL entities should be compared with one another. In this section we assign similarity measures to these pairs of RSL entities and include these similarity measures in the tables introduced in Section 5.5.

The section is structured as follows: Firstly, similarity measures are assigned to entities of the Terminology. Secondly, the elements of the DomainSpecification are discussed and finally the entities of the RequirementsSpecification are addressed. This order is necessary because the similarity of two Terms might be part of the similarity measure of two Phrases, i.e. the similarity of Terms is a local similarity measure that is used as part of another similarity measure.

6.6.1 Terminology

Terms contained in the Terminology are associated with a WordNetTerm. Both, the Term and the WordNetTerm contain a name (String) and a description (String). A basic similarity measure could only consider the name for the similarity evaluation. However, this would not solve the paraphrase problem. The paraphrase problem can be solved by evaluating the association to WordNetTerms. This can be achieved by several similarity approaches: Graph-based similarity, Description Logic and WordNet-Similarities (see below). A further possibility is to extend an Information Retrieval approach as described in Section 6.3. Which alternative is the most effective needs to be determined through experiments. All approaches have in common that they are based on the semantic relations defined in WordNet. Therefore, we refer to this local similarity measure as "WN-Sim" (for WordNet-based similarity).

Noun

A Noun contained in a query should be compared with the following entities: Actor, SystemElement, Notion, DomainStatement, NounPhrase, SimpleVerbPhrase and ComplexVerbPhrase (see Figure 6.5). All these RSL entities are associated with a Noun in the Terminology that is again associated with a WordNetTerm. The Similarity of these entities with a Noun is determined by the similarity of the associated WordNetTerms.

Exceptions are DomainStatement, SimpleVerbPhrase and ComplexVerbPhrase since they may contain one or two Nouns. If they contain two Nouns the determined similarities are combined.

Similarity (Noun, Actor / SystemElement / Notion / DomainStatement / NounPhrase / SimpleVerbPhrase / ComplexVerbPhrase) = WN-Sim1 (WordNetTerm, WordNetTerm)

Verb

A Verb contained in a query should be compared with the following entities: DomainStatement, SimpleVerbPhrase and ComplexVerbPhrase (see Figure 6.5). These RSL entities may be associated with a Verb in the Terminology that is again associated with a WordNetTerm. The similarity of these entities with a Verb is determined by the similarity of the associated WordNetTerms. An exception is the DomainStatement since not all DomainStatements have a Verb. If the DomainStatement does not contain a Verb the similarity is 0.

Similarity (Verb, DomainStatement / SimpleVerbPhrase / ComplexVerbPhrase) = WN-Sim2 (WordNetTerm, WordNetTerm)

Modifier

A Modifier contained in a query should be compared with NounPhrases (see Figure 6.5). A NounPhrase may be associated with a Modifier in the Terminology that is again associated with a WordNetTerm. The similarity of a NounPhrase with a Modifier is determined by the similarity of the associated WordNetTerms. However, not all NounPhrases are associated with a Modifier. In this case the similarity is 0.

Similarity (Modifier, NounPhrase) = WN-Sim3 (WordNetTerm, WordNetTerm)

Query / Case	Actor	SystemElement	Notion	DomainStatement	NounPhrase	SimpleVerbPhrase	ComplexVerbPhrase	UIElement	InputOutputDevice	InputOutputType	UIContainer	NaturalLanguageHypertextSentence
Terminology												
Noun	WN-Sim1	WN-Sim1	WN-Sim1	WN-Sim1	WN-Sim1	WN-Sim1	WN-Sim1	WN-Sim1	WN-Sim1	WN-Sim1	WN-Sim1	SimM4
Verb				WN-Sim2		WN-Sim2	WN-Sim2					SimM4
ModalVerb												
Determiner												
Modifier					WN-Sim3							SimM4
Preposition												
ConditionalConjunction												

Figure 6.5: RSL entities to be compared with Terms and assigned similarity measures.

6.6.2 Domain Specification

The similarity measures for DomainSpecification are described in the following and are summarised in Figure 6.5.

Phrases

All Phrases are contained in the DomainSpecification of a software case. The Phrases are associated with Terms in the Terminology via TermHyperlinks. These Terms are again associated with WordNetTerms. Phrases are well-structured RSL entities. Thus, the similarity may be evaluated with the graph-based similarity measure and the Description Logic similarity measure (possibly in combination) (G/DL-Sim). A further possibility is to determine the WordNet-based similarity for all Nouns, Verbs and Modifiers associated with a Phrase and combine these values. This alternative would not consider other words contained in a Sentence, e.g. Determiner, ConditionalConjunction, Preposition and ModalVerbs. Which alternative is the most effective one can only be determined through experiments.

Similarity (Phrase, Phrase) = G/DL-Sim(Phrase, Phrase)

Domain Statement

A DomainStatement contains a Phrase as name and may contain a textual description in one or more NaturalLanguageHypertextSentences. The attribute sentenceText contains this textual description (String) of a NaturalLanguageHypertextSentence. The similarity measure for DomainStatements should take into account both the textual description and the Phrase.

Similarity (DomainStatement, DomainStatement) = G/DL-Sim(Phrase, Phrase) + Set-of-Word-Sim (sentenceText, sentenceText)(SimM6)

Domain Elements

Actor and **SystemElement** are DomainElements. Both contain a Phrase as name and may contain NaturalLanguageHypertextSentences. The similarity measure should take into account the textual description as well as the Phrase.

Similarity (Actor, Actor) = G/DL-Sim(Phrase, Phrase) + Set-of-Word-Sim (sentenceText, sentenceText)(SimM6)

Similarity (SystemElement, SystemElement) = G/DL-Sim(Phrase, Phrase) + Set-of-Word-Sim (sentenceText, sentenceText)(SimM6)

A **Notion** is also a DomainElement. It contains:

- a Noun as name,
- several DomainStatements,
- several NotionAttributs and
- a textual description (NatualLanguageHypertextSentences)

A Notion may be connected with other Notions with a DomainElementAssociation and with a NotionSpecialisation. The similarity measure for Notions should take into account all parts of a Notion and its context, i.e. the related Notions. This can be achieved using the graph-based similarity measure and the Description Logic similarity measure (possibly in combination) (G/DL-Sim) for the structured part in combination with the Set-of-Word-Sim for the textual description (sentenceText).

Similarity (Notion, Notion) = G/DL-Sim(Notion, Notion) + Set-of-Word-Sim (sentenceText, sentenceText)(SimM7)

UIElement, InputOutputDevice, InputOutputType are specialisations of Notion. They can be compared using the same similarity measure as described for Notion. The InputOutputDevice should also be compared with SystemElement and vice versa. Both RSL entities contain a Noun as name. This Noun is linked with a WordNetTerm which can be used for a WordNet-based similarity as described in Section 6.6.1.

Similarity (InputOutputDevice, SystemElement) = WN-Sim1 (WordNetTerm, WordNetTerm)

Similarity (SystemElement, InputOutputDevice) = WN-Sim1 (WordNetTerm, WordNetTerm)

UIContainer is used to structure the UIElements contained in a DomainSpecification. Thus, a graph-based similarity measure and the Description Logic similarity measure can be used to compare the spatial or temporal order represented by UIContainers.

Similarity (UIContainer, UIContainer) = G/DL-Sim(UIContainer, UIContainer)

Domain Element Packages

DomainPackages contain DomainElements, which are potentially interconnected by DomainElementRelationship. These structures can be compared using a graph-based similarity measure and the Description Logic similarity measure (G/DL-Sim).

Domain Specification

A DomainSpecification contains several DomainElementPackages and one SystemElement that defines the system under development. Thus, the similarity of two DomainSpcifications is mainly defined by its packages and consequently the same similarity measure can be used as for packages.

Query / Case	**DomainSpecification**	ActorsPackage	Actor	SystemElementsPackage	SystemElement	NotionsPackage	Notion	DomainStatement	NounPhrase	SimpleVerbPhrase	ComplexVerbPhrase	UIElement	InputOutputDevice	InputOutputType	UIContainer	NaturalLanguageHypertextSentence
DomainSpecification	G/DL-Sim															
ActorsPackage		G/DL-Sim														SimM2
Actor			SimM6													SimM2
SystemElementsPackage				G/DL-Sim												SimM2
SystemElement					SimM6								WN-Sim1			SimM2
NotionsPackage						G/DL-Sim										SimM2
Notion							SimM7									SimM2
DomainStatement								SimM6								SimM2
NounPhrase									G/DL-Sim							SimM3
SimpleVerbPhrase									G/DL-Sim	G/DL-Sim	G/DL-Sim					SimM3
ComplexVerbPhrase									G/DL-Sim	G/DL-Sim	G/DL-Sim					SimM3
UIElement												SimM7				SimM2
InputOutputDevice					WN-Sim1								SimM7			SimM2
InputOutputType														SimM7		SimM2
UIContainer															G/DL-Sim	SimM2

Figure 6.6: The local similarity measures for RSL entities of the DomainSpecification are summarised in this table.

6.6.3 Requirements Specification

Natural Language Hypertext Sentence

A NaturalLanguageHypertextSentence contains a sentenceText (String) and may link Phrases. Even if parts of a NaturalLanguageHypertextSentence are linked to Phrases, some other parts might not be linked. In order to evaluate the whole sentence's text, IR should be applied to compare any RSL entity with a NaturalLanguageHypertextSentence. If a NaturalLanguageHypertextSentence links Phrases these should be evaluated additionally and both results should be combined. The following list specifies in detail which entities need to be evaluated when comparing NaturalLanguageHypertextSentences.

- Similarity (NaturalLanguageHypertextSentence, HyperlinkedSentence) = Set-of-Word-Sim (sentenceText, sentenceText) + Sim of linked Phrases (SimM1)
- Similarity (NaturalLanguageHypertextSentence, DomainElements / DomainElementPackages / DomainStatements) = Set-of-Word-Sim (sentenceText, text of DomainElementRepresentations) + Sim of contained Phrases (SimM2)
- Similarity (NaturalLanguageHypertextSentence, Phrases) = Set-of-Word-Sim (sentenceText, name of linked Terms) + Sim of Phrases (SimM3)
- Similarity (NaturalLanguageHypertextSentence, Terms) = Set-of-Word-Sim (sentenceText, name of Term) + Sim Term and Terms linked by NaturalLanguageHypertextSentence's Phrases. (SimM4)

SVO Sentence, Modal SVO Sentence and Conditional Sentence

SVOSentence, ModalSVOSentence and ConditionalSentence are well-structured Sentences. Each word is mapped to a Term in the Terminology and all Phrases are integrated in the DomainSpecification. The structure that relates to an SVOSentence is exemplified in Figure 4.9 on page 28. Evaluating the structure of these sentence types enables us to:

- identify whether a particular Actor is a Subject or Object in the Sentence,
- solve the paraphrase problem by evaluating the mapping to Terms in the Terminology,
- solve the ambiguity problem by evaluating the mapping to Terms in the Terminology.

The graph-based similarity measure and the Description Logic similarity measure may be used (possibly in combination) to determine the similarity of these sentences (G/DL-Sim).

PreconditionSentence, PostconditionsSentence, ConditionSentence, InvocationSentence and RejoinSentence

PreconditionSentence, PostconditionsSentence, ConditionSentence, InvocationSentence and RejoinSentence are not as well-structured as SVOSentence and the like. These Sentences may link Phrases but not every word needs to be linked to DomainSpecification and Terminology. Thus, the similarity measure needs to evaluate the sentence text.

Similarity (PreconditionSentences / PostconditionsSentences / ConditionSentences / InvocationSentences / RejoinSentences, PreconditionSentences / PostconditionsSentences / ConditionSentences / InvocationSentences / RejoinSentences) = Set-of-Word-Sim (sentenceText, sentenceText) + Sim of linked Phrases (SimM1).

NounPhrase & VerbPhrase of InteractionScenario

InteractionScenarios do not contain SVOSentences that aggregate Subject and Predicate. The similarity of SVOSentences, ModalSVOSentences and ConditionalSentences with NounPhrase and VerbPhrase of InteractionScenarios can be evaluated by comparing the NounPhrase with the Phrase linked by the Sentences Subject and the VerbPhrase with the Phrase linked by the sentences Predicate. The ConditionalSentence contains two Subjects and two Predicates. Thus, both need to be compared.

Similarity (NounPhrase + VerbPhrase (of InteractionScenarios), SVOSentences / ModalSVOSentences / ConditionalSentences) = Sim of NounPhrase and Phrases linked by Subject(s) + Sim of VerbPhrase and Phrases linked by Predicate(s) (SimM5).

RequirementRepresentation

A SentenceList consists of a set of sentences. The order of sentences in a SentenceList is not relevant. This is different for requirements representations that describe behavioural requirements, i.e. ConstraintLanguageScenarios and ActivityScenarios and InteractionScenarios, since they describe certain procedures that the system to be built should allow. A textual similarity

Query \ Case	NaturalLanguageHypertextSentence	SVOSentence	ModalSVOSentence	ConditionalSentence	PreconditionSentence	PostconditionSentence	ConditionSentence	InvocationSentence	RejoinSentence	NounPhrase (part of NounPhraseLifeline)	VerbPhrase (part of PredicateMessage)
NaturalLanguageHypertextSentence	SimM1	SimM1	SimM1	SimM1						SimM3	SimM3
SVOSentence	SimM1	G/DL-Sim	G/DL-Sim	G/DL-Sim						SimM5	SimM5
ModalSVOSentence	SimM1	G/DL-Sim	G/DL-Sim	G/DL-Sim						SimM5	SimM5
ConditionalSentence	SimM1	G/DL-Sim	G/DL-Sim	G/DL-Sim	SimM1	SimM1	SimM1			SimM5	SimM5
PreconditionSentence				SimM1	SimM1	SimM1	SimM1				
PostconditionSentence				SimM1	SimM1	SimM1	SimM1				
ConditionSentence				SimM1	SimM1	SimM1	SimM1				
InvocationSentence								SimM1			
RejoinSentence									SimM1		
NounPhrase (part of NounPhraseLifeline)	SimM3	SimM5	SimM5	SimM5						G/DL-Sim	
VerbPhrase (part of PredicateMessage)	SimM3	SimM5	SimM5	SimM5							G/DL-Sim

Figure 6.7: Similarity measures for comparing sentences.

measure would only compare the sf words / sf phrases but would not consider the order of described actions. Thus, the graph-based similarity measure and the Description Logic similarity measure are more appropriate for comparing these sf RequirementRepresentations.

Similarity (ConstraintLanguageScenario / ActivityScenario / InteractionScenario , ConstraintLanguageScenario / ActivityScenario / InteractionScenario) = G/DL-Sim (ConstraintLanguageScenario / ActivityScenario / InteractionScenario , ConstraintLanguageScenario / ActivityScenario / InteractionScenario)

A SentenceList may contain sentences of different types (see Figure 5.7 for an overview). Different types of sentences require different similarity measures (see Figure 6.7). Thus, a global similarity measure (similar to the set-of-sentences similarity measure, see Section 6.3.3) should summarise the similarities of the sentences contained in a SentenceList. The same holds if a SentenceList is compared with one of the RequirementRepresentations for behavioural requirements. These results are summarised in Figure 6.8 Similarity (SentenceList, SentenceList) = Similarity of contained HyperlinkedSentences (SimM8)

Similarity (SentenceList, ConstraintLanguageScenario / ActivityScenario / InteractionScenario) = Similarity of contained HyperlinkedSentences (SimM8)

Similarity (ConstraintLanguageScenario / ActivityScenario / InteractionScenario, SentenceList) = Similarity of contained HyperlinkedSentences (SimM8)

Query \Case	SentenceList	ConstraintLanguageScenario	ActivityScenario	InteractionScenario
SentenceList	SimM8	SimM8	SimM8	SimM8
ConstraintLanguageScenario	SimM8	G/DL-Sim	G/DL-Sim	G/DL-Sim
ActivityScenario	SimM8	G/DL-Sim	G/DL-Sim	G/DL-Sim
InteractionScenario	SimM8	G/DL-Sim	G/DL-Sim	G/DL-Sim

Figure 6.8: Similarity measures for comparing RequirementRepresentations.

RequirementSpecification

Requirements have different types (see Section 4.1 and can be associated with other Requirements using RequirementRelationships like Fulfills, Operationalises, Constrains, MakesPossible. Each Requirement has at least one RequirementRepresentation. These aspects should be considered when comparing Requirements.

A textual similarity measure would not consider the coherences defined by RequirementRelationships. This can be achieved using the graph-based similarity measure and the Description Logic similarity measure. These measures may produce poor results when comparing RequirementRepresentations that mainly contain NaturalLanguageHypertextSentences, however. Thus, a combination is needed. The RequirementRepresentations are compared according to Figure 6.8 while the relations between Requirements are compared using the graph-based approach. Thus, the similarity measures from Figure 6.8 are used as local similarity measures.

Additionally, it is necessary to compare a Requirement defined in the query with RequirementsPackages. This could be realised by pairwise comparison of the Requirement with all Requirements contained in the RequirementsPackage. However, this leads to poor results when the Requirement defined in the query covers several Requirements defined in the RequirementsPackage (comp. Requirement 7). Thus, comparison on the level of sentences is more appropriate. The resulting similarity measures are summarised in Figure 6.9.

Similarity (Requirement, Requirement) = Similarity of RequirementRepresentations combined with G-Sim of RequirementRelationships (SimM9)

Similarity (Requirement, RequirementsPackage) = Similarity of contained HyperlinkedSentences (SimM8)

RequirementsPackages contain a set of Requirements. The similarity of twoRequirementsPackages can be determined by the similarity of their Requirements. Thus, the similarity measure for Requirements is used as a local similarity measure. Furthermore, the RequirementsPackage should be compared with RequirementsSpecifications. This should be done on the level of Requirements rather than on the level of packages due to the fact that packages might differ in size.

Similarity (RequirementsPackage, RequirementsPackage) = Similarity of contained Requirements (SimM10)

Similarity (RequirementsPackage, RequirementsSpecification) = Similarity of contained Requirements (SimM10)

RequirementsSpecifications consist of several RequirementsPackages. The similarity of two RequirementsSpecifications can be determined by comparing the contained RequirementsPackages or contained Requirements. The latter would not take into account the structuring in packages. Which alternative leads to better results needs to be tested in experiments.

Similarity (RequirementsSpecification, RequirementsSpecification) = Similarity of contained RequirementsPackages (SimM10)

Query\Case	RequirementsSpecification	RequirementsPackage	Requirement
RequirementsSpecification	SimM10		
RequirementsPackage	SimM10	SimM10	
Requirement		SimM8	SimM9

Figure 6.9: Similarity measures for comparing Requirements, RequirementsPackages and RequirementsSpecifications.

Chapter 7

Similarity Measures - Design Considerations

When evaluating the effectiveness and efficiency of the similarity measures described in the previous chapters two prerequisites have to be fulfilled: first a reasonable amount of software cases have to be present, second the similarity measures have to be implemented in a retrieval framework. These prerequisites can only be fulfilled once the ReDSeeDS engine is (partly) implemented and the retrieval framework is integrated in Work Package WP5.

Architectural considerations for the retrieval framework are made in Deliverable D4.1.1. In the following, design considerations for integrating similarity measures are discussed and illustrated with preliminary examples.

7.1 Textual Similarity Measures - Design Considerations

Textual similarity measures will be implemented as extensions of existing textual similarity measures in RAISIN. All similarities will be computed online. If online computation is too slow, word similarities will be cached in tables or lists.

7.2 Description Logic Similarity Measures - Design Considerations

This section discusses aspects that may arise when Description Logics (DL) are used in the ReDSeeDS engine that will be developed in the Tasks 5.4 and 5.5. When using Description Logics for case retrieval the following design considerations have to be made:

1. Selection of a Description Logic inference tool
2. Interface between RSL and DL's internal representation
3. Representation of cases in DL
4. Representation of queries in DL
5. Services of DL used for retrieval
6. Similarity measures

In the following, these aspects are discussed and illustrated with examples.

7.2.1 Selection of a Description Logic Inference Tool

Currently we see two Description Logic inference tools that may be used within ReDSeeDS: Racer Pro[1] and the *Semantic Web Common Lisp Object System* (SWCLOS)[2]. While the former is a Description Logic tool based on tableau algorithms the latter uses meta-programming facilities to implement a DL reasoner. The first experiments that are described below use SWCLOS. However, when selecting an appropriate tool choice for ReDSeeDS (Deliverable D4.4 Repository Selection Report) other inference tools may be considered.

Properties of SWCLOS are:

- Semantic Web Processing in CLOS the Common Lisp Object System
- OWL Full (the *Web Ontology Language*[3]) implementation, i.e. classes are also considered as individuals

[1] www.racer-systems.com
[2] http://pegasus.agent.galaxy-express.co.jp/SemanticWeb-swclos-en.htm
[3] http://www.w3.org/TR/owl-ref/

- Satisfiability check and proactive entailment
- Handling of OWL Entailment Rules[4]
- Multiple classing
- Extended structural subsumption algorithm, which is incomplete for the existential restriction, but useful for OWL reasoning for most cases in practice [BCM$^+$03].
- Creator: Galaxy Express Corporation which offers a BSD-like Open Source Licence

For more details on SWCLOS we refer the interested reader to [KT06].

For our experiments we build an experimental environment consisting of Common Lisp, SWCLOS, RSL as OWL, cases as SWCLOS concepts and queries as SWCLOS individuals, which is explained in the following.

7.2.2 Interface between RSL and DL's Internal Representation

To evaluate the similarity measure based on Description Logics, RSL, which exists basically as UML diagrams from Enterprise Architect has to be transformed into a *Web Ontology Language* (*OWL*) ontology, which is in turn imported into SWCLOS (see Figure 7.1).

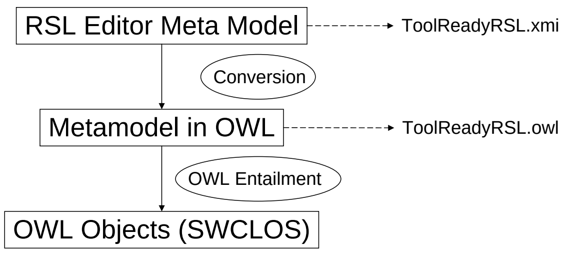

Figure 7.1: SWCLOS Information Flow.

This transformation is done by a conceptual and a format mapping.

Conceptual Mapping

To be able to make inferences about RSL in a Description Logic system, one has to convert the RSL meta model to a TBox. This means that all RSL entities (DomainSpecification, RequirementSpecification etc.) including their relationships (like NounLink etc.) have to be converted

[4]http://www.w3.org/TR/rdf-mt/

to concepts (*RSL concepts*) and roles (*RSL roles*) respectively. This is more or less a one to one mapping of meta classes to concepts. The relations like NounLink are represented as classes in ToolReadyRSL. Those *relation classes* have to be mapped to roles in DL.

Furthermore, when incorporating Terms from WordNet, these have to be mapped to concepts of a TBox too. Because WordNetTerms are usually *instances* of classes WN:Noun, WN:Verb etc. one has to shift those instances to corresponding concepts in the DL sense. Furthermore, between instances (terms) in WordNet the `hyponym-of` relation may be defined. This relation has to be mapped to the `is-a` relation between the corresponding concepts of the TBox.

Via the RSL relation `TRRSL:linksToWordNetEntry_d1e5622` the terms of the Terminology are linked to WordNet concepts, thus, the meaning of terms of the Terminology is finally established (see Figure 7.2 for an example in SWCLOS).

```
(defResource TY::order1 (rdf:type owl:Class)
  (owl:intersectionOf
   TRRSL:Noun
   (owl:Restriction (owl:onProperty TRRSL:isPartOfTerminology_d1e5647)
                    (owl:allValuesFrom TY::CommonTerminology))
   (owl:Restriction (owl:onProperty TRRSL:linksToWordNetEntry_d1e5622)
                    (owl:allValuesFrom a:|104902219-c|))))

(defResource TY::order2 (rdf:type owl:Class)
  (owl:intersectionOf
   TRRSL:Noun
   (owl:Restriction (owl:onProperty TRRSL:isPartOfTerminology_d1e5647)
                    (owl:allValuesFrom TY::CommonTerminology))
   (owl:Restriction (owl:onProperty TRRSL:linksToWordNetEntry_d1e5622)
                    (owl:allValuesFrom a:|106246680-c|))))
```

Figure 7.2: We show here the SWCLOS notation of TBox concepts. Concepts of the Terminology are related to concepts of WordNet by using the relation `TRRSL:linksToWordNetEntry_d1e5622`. This relation finally establishes the meaning of `order1` and `order2` in Figure 6.2 (see Section 6.4 on page 54). `a:|104902219-c|` denotes the concept of the WordNet Synset 104902219, which is the set ("order" "purchase order"). `a:|106246680-c|` denotes the concept of the WordNet Synset 106246680, which is the set ("order" "ordering").

Format Mapping

Typical Description Logic inference systems import ontologies written in the *Web Ontology Language* (*OWL*). The Web Ontology Language OWL is a semantic markup language for publishing and sharing ontologies on the World Wide Web. OWL is developed as a vocabulary extension of RDF (the Resource Description Framework) and is derived from the DAML+OIL[5] Web Ontology Language.

[5]http://www.w3.org/TR/daml+oil-reference

OWL is designed for use by applications that need to process the content of information instead of just presenting information to humans. OWL facilitates greater machine interpretability of Web content than that supported by XML, RDF, and RDF Schema (RDF-S) by providing additional vocabulary along with a formal semantics. OWL has three increasingly-expressive sublanguages: OWL Lite, OWL DL, and OWL Full.[6]

To be able to use RSL in a Description Logic system, RSL has to be converted to OWL. Currently there exists a conversion tool written by University of Koblenz which converts the ToolReadyRSL metamodel (see Chapter 1) to OWL concepts.

RSL is based on UML 2.0. OWL Full and UML 2.0 are two modelling languages that have a lot of features in common. The basic modelling elements can easily be transformed from UML 2.0 classes to OWL Full classes [HEC+04]. Next to the notion of classes, both OWL Full and UML 2.0 share the notions of instances of classes (OWL: individual), attributes (OWL: property with basic data type), associations (OWL: property with associated class type), generalisation (OWL: `subclass` or `intersection`), multiplicity (OWL: `minCardinality`, `maxCardinality`, `inverseOf`), enumerations (OWL: `oneOf`), (non-)navigables (OWL: domain, range), packages (OWL: ontology) and dependencies (OWL: reserved name).

Some modelling elements, however, are not common for both languages. While OWL Full offers intersection, union and complement, and relies on the unique name assumption (UNA) and equivalence of classes, UML 2.0 does not. On the other hand, UML 2.0 offers behavioural features (i.e. static operations, interfaces and abstract classes), complex objects (i.e. composition and aggregation), as well as ports and connectors which OWL Full does not offer.

However, for the ToolReadyRSL metamodel a mapping can be given. As an example we give in Figure 7.3 the representation of DeterminerLink in OWL.

OWL representations are imported via SWCLOS into the experimental environment.

[6]http://www.w3.org/TR/owl-features/

```
<owl:Class rdf:ID="Determiner">
    <rdfs:subClassOf rdf:resource="#Term"/>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#determinerLink-of"/>
            <owl:minCardinality
               rdf:datatype="http://www.w3.org/2001/XMLSchema#nonNegativeInteger">0</owl:minCardinality>
            <owl:maxCardinality
               rdf:datatype="http://www.w3.org/2001/XMLSchema#nonNegativeInteger">2147483647</owl:maxCardinality>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#determinerSpecialisationRelation"/>
            <owl:minCardinality
               rdf:datatype="http://www.w3.org/2001/XMLSchema#nonNegativeInteger">0</owl:minCardinality>
            <owl:maxCardinality
               rdf:datatype="http://www.w3.org/2001/XMLSchema#nonNegativeInteger">2147483647</owl:maxCardinality>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty
               rdf:resource="#determinerSpecialisationRelation-of"/>
            <owl:minCardinality
               rdf:datatype="http://www.w3.org/2001/XMLSchema#nonNegativeInteger">0</owl:minCardinality>
            <owl:maxCardinality
               rdf:datatype="http://www.w3.org/2001/XMLSchema#nonNegativeInteger">2147483647</owl:maxCardinality>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
```

Figure 7.3: Representing RSL with OWL.

7.2.3 Representation of Software Cases in DL

As described in Section 6.4 software cases are represented with concepts of the TBox. As an example parts of two software cases depicted in Figure 6.2 on page 54 are shown in Figure 7.4.

```
(defResource c1::Client--NounPhrase (rdf:type owl:Class)
  (owl:intersectionOf
   TRRSL:NounPhrase
   (owl:Restriction (owl:onProperty TRRSL:nounLink)
                    (owl:allValuesFrom TY::client))
   (owl:Restriction (owl:onProperty TRRSL:isPartOfActor_d1e4112)
                    (owl:allValuesFrom c1::Client))))

(defResource c1::order--NounPhrase (rdf:type owl:Class)
  (owl:intersectionOf
   TRRSL:NounPhrase
   (owl:Restriction (owl:onProperty TRRSL:nounLink)
                    (owl:allValuesFrom TY::order1))
   (owl:Restriction (owl:onProperty TRRSL:isPartOfDomainStatement_d1e3741)
                    (owl:allValuesFrom c1::order--DomainStatement))))

(defResource c2::Customer--NounPhrase (rdf:type owl:Class)
  (owl:intersectionOf
   TRRSL:NounPhrase
   (owl:Restriction (owl:onProperty TRRSL:nounLink)
                    (owl:allValuesFrom TY::customer))
   (owl:Restriction (owl:onProperty TRRSL:isPartOfActor_d1e4112)
                    (owl:allValuesFrom c2::Customer))))

(defResource c2::order--NounPhrase (rdf:type owl:Class)
  (owl:intersectionOf
   TRRSL:NounPhrase
   (owl:Restriction (owl:onProperty TRRSL:nounLink)
                    (owl:allValuesFrom TY::order2))
   (owl:Restriction (owl:onProperty TRRSL:isPartOfDomainStatement_d1e3741)
                    (owl:allValuesFrom c2::order--DomainStatement))))
```

Figure 7.4: The crucial parts of the two SVO Sentences shown in Figure 6.2 as concepts written in SWCLOS, i.e. "client", "customer", and two meanings of "order". `c1` and `c2` denote Case 1 and Case 2. `TY` refers to the Terminology, which specifies the meaning of the NounPhrases (see Figure 7.2).

Thus, all RSL entities of a software case specialise RSL concepts, like `c2::order-NounPhrase` specialises `TRRSL:NounPhrase`. All known cases are specified in this way. The entailment rules of DL compute automatically the specialisation relations between the RSL entities of cases. If a new software case is added to the fact repository, it has to be mapped to concepts like those shown in Figure 7.4.

7.2.4 Representation of Queries in DL

A query is represented by instances of RSL concepts. In Figure 7.5 one query is shown which directly corresponds to the NounPhrase of Case 2. Because all roles of an individual are checked during instance recognition, all roles have to have appropriate individuals as role fillers. This is why the whole instance chain from `qu::myTy-Order2`, which specifies the intended meaning by `(TRRSL:linksToWordNetEntry_d1e5622 qu::mywn-meaning)`, to `qu::my-terminology` has to be instantiated.

Note that the instances in the query are instances of RSL concepts not of case concepts like `c2::Customer-NounPhrase`. This means, that the query itself is not yet related to cases, which is appropriate.

```
(defIndividual qu::my-terminology (rdf:type TY::CommonTerminology))

(defIndividual qu::mywn-meaning (rdf:type a:|106246680-c|))

(defIndividual qu::my-DomainSpecification
                (rdf:type TRRSL:DomainSpecification))

(defIndividual qu::my-NotionsPackage (rdf:type TRRSL:NotionsPackage)
  (TRRSL:isPartOfDomainSpecification_d1e3573 qu::my-DomainSpecification))

(defIndividual qu::my-Order (rdf:type TRRSL:Notion)
  (TRRSL:isPartOfNotionsPackage_d1e3772 qu::my-NotionsPackage))

(defIndividual qu::my-OrderDomainStatement
                (rdf:type TRRSL:DomainStatement)
  (TRRSL:isPartOfNotion_d1e3666 qu::my-Order))


(defIndividual qu::myTy-Order2 (rdf:type TY::order2)
  (TRRSL:isPartOfTerminology_d1e5647 qu::my-terminology)
  (TRRSL:linksToWordNetEntry_d1e5622_WN qu::mywn-meaning))


(defIndividual qu::myOrder--NounPhrase (rdf:type TRRSL:NounPhrase)
  (TRRSL:nounLink qu::myTy-Order2)
  (TRRSL:isPartOfDomainStatement_d1e3741 qu::my-OrderDomainStatement))
```

Figure 7.5: A query represented as instances in SWCLOS notation. Further queries may be represented which use other meanings of "order" or "client". Here the query represents a NounPhrase which contains "order" by instantiating the concept `a:|106246680-c|`.

7.2.5 Services of DL used for Retrieval

The main service of DL used for case retrieval is the instance recognition service (see Section 6.4). As the query is seen as an instance structure, their relations to potential concepts can be computed by instance recognition. Currently in our experiments this has to be done by comparing the query to every case concept. In SWCLOS the instance recognition service is implemented by `typep` see Figure 7.6.

```
(typep qu::myOrder--NounPhrase c2::order--NounPhrase) --> t
(typep qu::myOrder--NounPhrase c1::order--NounPhrase) --> nil
```

Figure 7.6: Checking the query against the corresponding part of two cases and the expected result.

7.2.6 Similarity Measures

The taxonomic similarity measure sketched in Section 6.4 is not yet implemented, but the example in this section describes its usage.

7.2.7 Summary

The following observations should be noted, when applying DL to RSL case retrieval:

- During instance recognition the final WordNet meaning determines the concept an instance belongs to.
- Without some similarity measure a DL retrieval corresponds to a boolean information retrieval method, which determines if a query is similar to a case or not.
- All roles in a query have to have appropriate individuals as fillers.
- An advantage of DL is given when the query is more specific than the cases (e.g. Query: "commercial document" Case: "Document"). Then the DL services would identify those more general cases, by relating the query to the corresponding concepts.
- The main advantage of DL would be obtained, if the concepts of WordNet were described by roles, not only by a string set. If a query were to use such roles for an instance of the top concept `Thing` the DL services would infer what specific concept is meant by the query.
- Currently in RSL variability occurs in diverse places. Notions can for example be described by specialisations and relations between them or SentenceLists can have a variable number of SVOSentences as parts. In each such case where variability in structure may occur, DL services would bring advantages in comparing the structure of a case with a query, because they can compute similarity measures based on structure.

7.3 Graph-based similarity - Design Considerations

In order to use the SiDiff algorithm depicted in section 6.5, the SiDiff tool can be used. This section describes the SiDiff tool as well as the details of the similarity measure implemented in this tool.

7.3.1 The SiDiff Tool

The SiDiff tool is developed by the software engineering group at the University of Siegen, Germany under leadership of Professor Kelter. The SiDiff developers state that their tool is able to produce more exact results than other graph-based tools. They explain this with the fact that SiDiff takes the semantics of model elements into account.

To evaluate the similarity of two SCL models with the SiDiff tool, the SCL models stored in the fact repository are exported to a format that the SiDiff tool can process. Currently, this is an XML-file.

The result of the similarity calculation is a table which contains similar elements in both models and their similarity. The total similarity of the two models can then be calculated depending on the number of similar elements, their similarity and the number of elements without a matching partner.

7.3.2 Similarity Measure

The quality of the similarity calculated by the SiDiff tool depends on several parameters. For each element type, two elements of that type are compared using the elements properties. Since the impact of several properties may differ from type to type and there may exist properties that are not defined for every element type, for each type it can be specified in which way each property affects the similarity of two elements [Weh04]. For instance, in class models the similarity of two classes may depend on the following aspects:

- Similar names: The similarity of two elements is higher if their names are equal. Further, it is also true that only small changes in the name lead to a higher similarity than totally different names.

- Matching parents: If the parents of both elements match, the similarity of the elements is probably higher than without matching parents. This is one of the key attributes for the top-down phase of the SiDiff algorithm. See chapter 6.5.2 for further information on that algorithm.
- Attributes: As well as the name, the attributes of the elements may have significant influence on the similarity.
- References: As the last property, any references to other elements are important for the similarity of two elements. This is similar to matching parents.

For each of these properties and each of the element types, a factor between 0 and 1 can be specified. If the factor is set to 0, the property has no impact on the similarity for that element type, if it is set to 1, it has a high influence. As an example, one may decide that the similarity of names is the most important factor and thus weight the name with a 0.5. Further, the impact of matching parents and references may be equal, hence both are weighted with a 0.2. As the last component, the attributes may have a low influence so their factor is set to 0.1. In this case, the sum of the factors is 1.0, but in general, the factors are normalized anyway so that their sum is 1.0.

In addition to these factors for the several properties, the similarity threshold (see also Section 6.5.2) also has a great influence on the accuracy of the SiDiff algorithm.

To specify reasonable values for the factors as well as the similarity threshold, experiments are needed as soon as the software cases are available in the project.

Chapter 8

Conclusion

This deliverable describes the concepts behind the software case similarity measure that will be part of the ReDSeeDS Engine. Thus, this deliverable provides important input for Work Package 5 (Development of the ReDSeeDS system prototype). The main contributions of this deliverable are:

- An overview of similarity measures that are relevant for comparing RSL requirement models and a brief introduction of the corresponding research areas (see Chapter 3).
- A detailed specification of requirements to be met by the similarity measure (see Chapter 5).
- A concept for a similarity measure for RSL requirement models. This concept is based on the requirements and combines the similarity measures introduced in Chapter 3 appropriately (see Chapter 6).
- An overview of aspects that need to be considered when implementing the similarity measures as part of the ReDSeeDS Engine in Work Package 5 (see Chapter 7).
- A number of questions that need to be considered during the experiments (see Chapter 6 and 7).

This deliverable focuses on the combination of similarity measures from different research areas. This takes into account the different aspects of RSL requirements models. RSL integrates textual descriptions and model-based descriptions in one language. Thus, the similarity measure needs to combine similarity measures for these different types of artefacts.

Several similarity measures for reuse of software development artefacts have been developed. These similarity measures take different types of software development artefacts into account. Most approaches focus on artefacts produced in later stages of the development cycle, e.g. design artefacts or even code. Work that focuses on requirements artefacts typically relies on one type of similarity measure only, usually Information Retrieval. Our work, however, combines measures from different research areas.

Further work on the ReDSeeDS software case similarity measure involves reconciling the similarity measure requirements with the overall requirements on the ReDSeeDS engine (Task 5.2 Specification of user requirements for the ReDSeeDS Engine Prototype). In cases of conflict this might require the adaptation of similarity measure requirements.

Experiments are needed to further improve the similarity measure. To conduct such experiments many RSL-consistent requirements models are needed. In Information Retrieval, huge document collections are usually needed to assess the effectiveness of similarity measures. Thus, these experiments cannot be conducted until tool-support for RSL is available.

Bibliography

[ABS96] Klaus-Dieter Althoff and Brigitte Bartsch-Spörl. Decision support for case-based applications. *Wirtschaftsinformatik*, 38(1):8–16, February 1996.

[AP94] A Aamodt and E Plaza. Case-based reasoning: Foundational issues, methodological variations, and system approaches. *Artificial Intelligence Communications*, 7(1):39–59, 1994.

[BBG$^+$99] Ralph Bergmann, Sean Breen, Mehmet Göker, Michel Manago, and Stefan Wess. *Developing Industrial Case-Based Reasoning Applications – The INRECA Methodology*. Springer Verlag, 1999.

[BBMW99] Ralph Bergmann, Sean Breen, Michel Manago, and Stefan Wess. *Developing Industrial Case-Based Reasoning Applications - The INRECA Methodology*. Springer, 1999.

[BCM$^+$03] F Baader, D Calvanese, D McGuinness, D Nardi, and P Patel-Schneider. *The Description Logic Handbook*. Cambridge University Press, 2003.

[Ber98] Ralph Bergmann. On the use of taxonomies for representing case features and local similarity measures. In *6th German Workshop on CBR*, 1998.

[BR91] V Basili and H-D Rombach. Support for comprehensive reuse. *IEEE Software Engineering Journal*, 6(5):303–316, 1991.

[BS85] Ron J. Brachman and James G. Schmolze. An Overview of the KL-ONE Knowledge Representation System. *Cognitive Science*, 9(2):171–216, 1985.

[BYRN99] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval*. ACM Press, New York, 1999.

[GCGADA99] Pedro A. González-Calero, Mercedes Gómez-Albarran, and Belén Díaz-Agudo. Applying dls for retrieval in case-based reasoning. In *Proc. of the 1999 International Workshop on Description Logics (DL'99)*, 1999.

[GF04] David A. Grossman and Ophir Frieder. *Information retrieval: algorithms and heuristics*. Springer, 2004.

[Gom04] Paulo Gomes. Software design retrieval using bayesian networks and WordNet. *Lecture Notes in Computer Science*, 3155:184–197, 2004.

[GPP+04] Paulo Gomes, Francisco C. Pereira, Paulo Paiva, Nuno Seco, Paulo Carreiro, José L. Ferreira, and Carlos Bento. Using wordnet for case-based retrieval of uml models. *AI Commun.*, 17:13–23, 2004.

[HEC+04] L. Hart, P. Emery, B. Colomb, K. Raymond, S. Taraporewalla, D. Chang, Y. Ye, E. Kendall, and M. Dutra. Owl full and uml 2.0 compared. www.itee.uq.edu.au/ colomb/Papers/UML-OWLont04.03.01.pdf, March 2004.

[Hun76] M. Douglas Hunt, James W. und McIlroy. An algorithm for differential file comparison. Computing Science Technical Report 41, Bell Laboratories, June 1976.

[Kol93] Janet Kolodner. *Case-Based Reasoning*. Morgan Kaufmann, 1993.

[KT06] S. Koide and H. Takeda. Owl-full reasoning from an object oriented perspective. In R. Mizoguchi, Z. Shi, and F. Giunchiglia, editors, *The Semantic Web (ASWC 2006)*, volume 4185 of *Lecture Notes in Computer Science*, pages 263–277. Springer Verlag, 2006.

[Kur04] Dominik Kuropka. *Modelle zur Repräsentation natürlichsprachlicher Dokumente - Ontologie-basiertes Information-Filtering and -Retrieval mit relationalen Datenbanken*. Logos, 2004.

[LBSBW98] Mario Lenz, Brigitte Bartsch-SpÃ¶rl, Hans-Dieter Brukhard, and Stefan Wess, editors. *Case-Based Reasoning Technology - From Foundations to Applications*, chapter Textual CBR, pages 115–137. Springer, 1998.

[LHK98] Mario Lenz, André Hübner, and Mirjam Kunze. Textual CBR and information retrieval a comparison. *Lecture Notes in Computer Science*, 1400, 1998.

[LYM05] Shuang Liu, Clement Yu, and Weiyi Meng. Word sense disambiguation in queries. In *CIKM '05: Proceedings of the 14th ACM international conference on Information and knowledge management*, pages 525–532, New York, NY, USA, 2005. ACM Press.

[MBF+90] George A Miller, Richard Beckwith, Christiane Fellbaum, Derek Gross, and Katherine J Miller. Introduction to WordNet: An on-line lexical database. *International Journal of Lexicography*, 3(4):235–244, 1990.

[MGMR02] Sergey Melnik, Hector Garcia-Molina, and Erhard Rahm. Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In *ICDE*, pages 117–128, 2002.

[Min74] Marvin A. Minsky. A framework for representing knowledge. Technical report, Artificial Intelligence Memo 306, Massachusetts Institute of Technology, Cambridge, MA, USA, 1974.

[NAB07] Markus Nick, Klaus-Dieter Althoff, and Ralph Bergmann. Experience management. *Kuenstliche Intelligenz*, 2:48–49, March 2007.

[Nic05] Markus Nick. *Experience Maintenance through Closed-Loop Feedback*. PhD thesis, University of Kaiserslautern, Germany, 2005. Published by Fraunhofer IRB Verlag, Germany, ISBN 3-8167-6927-6.

[PPM04] T. Pedersen, S. Patwardhan, and J. Michelizzi. Wordnet::similarity - measuring the relatedness of concepts. In *In Proceedings of the Nineteenth National Conference on Artificial Intelligence (AAAI-04)*, 2004.

[Qui69] R. Quillian. Semantic memory. In Marvin Minsky, editor, *Semantic Information Processing*. MIT Press, 1969.

[Sin01] Amit Singhal. Modern information retrieval: A brief overview. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 2001.

[SM83] Gerard Salton and Michael J. McGill. *Introduction to Modern Information Retrieval*, chapter Retrieval Evaluation, pages 157–198. McGraw Hill, 1983.

[TA97] C Tautz and K-D Althoff. Using case-based reasoning for reusing software knowledge. *Lecture Notes in Computer Science*, 1266:156–165, 1997.

[TA98] Carsten Tautz and Klaus-Dieter Althoff. Operationalizing comprehensive software knowledge reuse based on CBR methods. In Lothar Gierl and Mario Lenz, editors, *6th German Workshop on CBR*, volume 7 of *IMIB Series*, pages 89–98, Berlin, Germany, March 1998. Institut für Medizinische Informatik und Biometrik, Universität Rostock.

[TG98] Carsten Tautz and Christiane Gresse von Wangenheim. REFSENO: A representation formalism for software engineering ontologies. Technical Report IESE-Report No. 015.98/E, Fraunhofer Institute for Experimental Software Engineering, Kaiserslautern (Germany), 1998.

[UK05] Jörg Niere Udo Kelter, Jürgen Wehren. A generic difference algorithm for uml models. In *Proceedings of the SE 2005, Essen, Germany*, Essen, Germany, March 2005.

[VVR+05] Giannis Varelas, Epimenidis Voutsakis, Paraskevi Raftopoulou, Euripides G.M. Petrakis, and Evangelos E. Milios. Semantic similarity methods in WordNet and their application to information retrieval on the web. In *WIDM '05: Proceedings of the 7th annual ACM international workshop on Web information and data management*, pages 10–16, New York, NY, USA, 2005. ACM Press.

[Weh04] Jürgen Wehren. Ein XMI-basiertes Differenzwerkzeug für UML Diagramme. Diploma thesis, University of Siegen, Germany, 2004.