Analogical Reuse of Object-Oriented Analysis Models

Solveig Bjørnestad

Dissertation for the degree Doctor Rerum Politicarum (dr. polit.)

University of Bergen

2008

ISBN 978-82-308-0574-9
Bergen, Norway 2008

Printed by Allkopi Ph: +47 55 54 49 40

Analogical Reuse of Object-Oriented Analysis Models

Solveig Bjørnestad

Dissertation for the degree Doctor Rerum Politicarum (dr. polit.)

at the University of Bergen

Department of Information Science and Media Studies

The faculty of Social Science

2008

Acknowledgments

This dissertation is the result of my work within the ROSA (Reuse of Object-oriented Specifications through Analogy) project. Several people have been part of this project over the years, but I will particularly thank my colleague and supervisor Bjørnar Tessem. He has been an important inspirator and we have shared many discussions over the topics covered in this thesis.

I will also like to thank my other colleagues at the department for supporting me. Particularly I want to mention Jon-Helge Knudsen who has given me valuable advice related to the statistical tests. My thanks also go to Weiqin Chen, Richard E. Moe, and Andreas L. Opdahl who have read and commented versions of the thesis and given valuable feedback. Mike Spector, a friend and previous colleague, is thanked because he at a critical point of time stepped in and made me move on.

Finally I would like to thank my children and my mother, for all support over these years, and last, but not least, Finn who have tolerated all my ups and downs over the last hectic period of writing.

SOLVEIG BJØRNESTAD

The University of Bergen

December 2007

Analogical Reuse of Object-Oriented Analysis Models

Solveig Bjørnestad

Software reuse involves using again software artifacts that have been successfully built before. To be successful with software reuse, techniques for reuse must be integrated into both the information system development process and the programming environment. If potential reuse can be identified early in an information system development process, the gain in development time can be substantial. Techniques to automatically identify reuse candidates incorporated in software development tools would increase the benefits for software development even more.

In this dissertation the incorporation of reuse techniques based on analogical reasoning (AR) in tools for software development is proposed. These techniques use information about the structure and semantics of a model from the analysis of a software system to try to identify potential analogous models.

Analogical reasoning is typically described as consisting of a set of phases. Although other AR phases are equally important, the focus of this thesis is on the retrieval and mapping phases of AR. The proposed approach is demonstrated using OOram *role models*. OOram is an object-oriented modelling notation resembling UML sequence diagrams. OOram models were chosen because they focus entirely on the analysis of a problem and does not take into consideration what objects will play the various roles in the system. The findings in this thesis are applicable also for such models.

A user creates a *role model* during the analysis of a new project. To prevent too much detailed work at this stage, it would be advantageous if a tool could support the process by identifying reusable candidates from a software development repository. The proposed approach implements support for a tool that can search a repository for models that are analogous to the model being created. The user must then evaluate the identified models to see if they are suitable within the project.

AR is used to identify similar cases from different problem domains. A *similarity model* for OOram role models that uses a combination of structural and semantic information about the models to identify similarities is proposed. At the time the ROSA project was initiated, this was a natural choice.

The requirements of the similarity model are that it is able to distinguish potentially useful models from the ones that cannot be reused. In the approach suggested in this thesis, each named component in the model repository is linked to a *word meaning* in a term space. This term space is modelled after WordNet, an electronic, lexical database.

During *retrieval*, information about structure and semantics of the models is used. All new role models are given a structure *description* before they are stored in the repository. This information is, during retrieval, used as an index. Semantic similarity among models is during retrieval found by identifying distance in the semantic network. An upper bound for the semantic similarity between a *target model* and each of the *base models* in the repository is identified, and this result is combined with a structural similarity, based on the structure descriptions, to form a retrieval similarity.

During the *mapping phase*, the most promising base models after retrieval are compared to the target model. Mapping between a target and each of the retrieved base models is done using a genetic algorithm that tries to optimize the mapping between the two models based on their structure and semantics, resulting in a mapping similarity. The balance between semantics and structure in the similarity model is vital both during retrieval and mapping.

Experiments are described in which analogies are identified between a target model and the models in a repository containing 133 models. In this context a good analogy for a role model is a role model for which we calculate a high mapping similarity. This implies that the models have similar structure, and roles that are positioned at comparable positions in the structures have similar semantics.

In 21 of 24 cases, the model with the highest mapping similarity is identified from among the top 30 ranked models during retrieval. Experiments also show that if considering the 5 highest ranked models according to mapping similarity in each of the 24 cases, more than 85 % of them will be localized among the top 30 ranked models after retrieval. The findings reported show that the suggested approach is viable, although further studies are necessary. The top ranked model may prove not to be the best analogy after further analysis. The user must evaluate the mappings.

Contents

List of Tables

List of Figures

Acronyms

ACME	**A**nalogical **C**onstraint **M**apping **E**ngine
AI	**A**rtificial **I**ntelligence
AIR	**A**dvisor for **I**ntelligent **R**euse
ANN	**A**rtificial **N**eural **N**ets
AR	**A**nalogical **R**easoning
COM	**C**omponent **O**bject **M**odel—from Microsoft.
CBR	**C**ase **B**ased **R**easoning
CBSE	**C**omponent-**B**ased **S**oftware **E**ngineering
CORBA	**C**ommon **O**bject **R**equest **B**roker **A**rchitecture
COTS	**C**ommercial **O**f-**T**he-**S**helf software
DFD	**D**ata **F**low **D**iagram
DTD	**D**ocument **T**ype **D**efinition
ERD	**E**entity **R**elationship **D**iagram
IDE	**I**ntegrated **D**evelopment **E**nvironment
IDL	**I**ntegrated **D**efinition **L**anguage
GA	**G**enetic **A**lgorithms
GUI	**G**raphical **U**ser **I**nterface
IC	**I**ntegrated **C**ircuit
IR	**I**nformation **R**etrieval
IS	**I**nformation **S**ystem
NN	**N**eural **N**et
ODBMS	**O**bject-oriented **D**atabase **M**anagement **S**ystems
OO	**O**bject **O**rientation
OMG	**O**bject **M**anaging **G**roup
OOram	**O**bject **O**riented **r**ole **a**nalysis **m**odeling
ROSA	**R**euse of **O**bject-oriented **S**pecifications through **A**nalogy
SE	**S**oftware **E**ngineering
SME	**S**tructure **M**apping **E**ngine
STL	The **S**tandard **T**emplate **L**ibrary—has become a standard for C++
UML	The **U**nified **M**odeling **L**anguage—industry-standard from **OMG**
XML	e**X**stensible **M**arkup **L**anguage

Chapter 1

Introduction

The problem of software reuse has received much attention during the last few decades, see for example Johnson and Foote (1988), Biggerstaff and Richter (1989), Rosson and Carrol (1990), Maiden and Sutcliffe (1992), Tracz (1994b), and Heumesser and Houdek (2003). Main incentives for this effort have been increasing software development expenses, delayed delivery, and unsatisfactory software quality. Software developers who want to create software products of high quality in a cost effective way must somehow build upon previous experience and work. Suggested solutions have been concerned with technical, process-oriented or organizational aspects of reuse.

This chapter starts by motivating my interest in software reuse, and goes on to give an outline and scope of the ROSA project where analogical reasoning is applied to support reuse of analysis models. A scenario is given to indicate how the envisioned tool support for reuse by analogy can be realized. There follows a discussion of the research problem, a description of the research design and an overview of the thesis and the outline of research contributions. In this thesis, *software reuse* in general, refers to reuse of any type of asset or artifact from the software development process that can be reused either as is or in a modified form. The term *software artifact* is used to represent any such component.

1.1 Motivation

The earliest attempts of software reuse focused on language constructs and construction of libraries of general, reusable functions. Such attempts were successful within narrow domains such as graphics and mathematics. In many other domains, less specific and formal, a similar approach turned out to be much more difficult. Solutions were searched for elsewhere, and object-oriented languages and software frameworks were among the suggested solutions. However, although valuable contributions, they were not the solution of the software crises.

Tracz (1988) claimed that software reuse had not evolved far beyond its most rudimentary forms of subroutine, or class, libraries and brute-force modification. While at that time claiming that artificial intelligence (AI) had not contributed to software reuse, Tracz (1994a) argued that “software reuse is the common ground where AI and software engineering will meet”. This was attributed to the strength of AI within knowledge acquisition and representation experience. When trying to reuse artifacts from earlier stages of the software development process, i.e., prior to coding, AI is considered particularly important.

It has long been realized that software reuse must be planned, and that it must be integrated into the software development process itself. Schmidt (1999) stated that although *opportunistic* reuse takes place when software developers cut and paste code previously developed by themselves or by others within a small group, this type of reuse does not scale up for large projects and widespread use. This field has received increased interest in recent years, e.g., Conradi (1996), Lam, Jones, and Britton (1998), Sutcliffe (2000), Frakes (2002), and Sherif, Appan, and Lin (2006). The goal is to achieve *systematic* software reuse, and this requires both organizational and technical changes.

Large scale reuse can only be achieved if a systematic reuse process is established. This includes reuse from early phases such as requirements engineering in addition to code reuse (Lam et al., 1998). Organizations that develop applications within one domain may have a particularly good reuse potential. It is easy to accept that models can be reusable within a given domain, because many of the components may be similar, and often the way that these components are interconnected will also be the same. However, a lot may also be gained by applying reuse across domains. Sutcliffe (2000) gives an example of how generic domain models can be reused as templates across domains. The use of analysis patterns is another area where analysis reuse can be accomplished across domains (Fowler, 1997).

It is important both to make available reusable assets, create incentives in the organization to make use of them, and make available and motivate people to use tools that can help reuse to take place. To develop domain models for any given domain is a demanding and time consuming task. One of the suggested ways of creating domain models is by studying existing systems and extracting the domain knowledge from them (Neighbors, 1989).

In recent years, focus has shifted more and more towards component-based development. Several definitions exist of what a component is, but an often-used definition is that of Szyperski et al. (2002), where a component is a unit of composition deployed independently of the product. One effect of this shift is that the developer has less control over the architecture and design of a new system because so many design decisions are already taken by component developers. The developer's design decisions are now shifted more in direction of selecting the right components, in contrast to a situation where you design your own components from scratch. In this last situation, Wallnau, Hissam, and Seacord (2002) claim that decisions about the design of the components' interfaces are more important.

If available components do not exactly match the requirements for the system, these requirements, according to Conradi (1996), may have to be modified or relaxed. To make software design less tedious when reusable components are involved, it will be advantageous to identify the reuse potential prior to the design stage. Thus previous successful designs can be taken into use as reusable components. Cechich and Piattini (2007) suggest an approach of identifying candidate components where functionality evaluation drive the analysis to exclude candidate components so that the best alternative remains.

1.2 The ROSA Approach to Software Reuse

The ROSA (Reuse of Object-oriented Specifications through Analogy) project was initiated by Tessem and Bjørnestad in 1994 and ran at the Department for Information Science at University of Bergen through 2006 (Tessem, Bjørnestad, Tornes, and Steine-Eriksen, 1994; Tessem and Bjørnestad, 1997). So far the project has produced four Master theses (Tornes, 1995; Steine-Eriksen, 1995; Ul-Haq, 1997; Midttun, 1998) and one PhD thesis (Ellingsen, 1997a). Several of these projects have studied the use of individual techniques for used for mapping in isolated projects. The overall goal of the project was to study how identification of reuse candidates could be done early in a software project, as this may reduce the project's use of resources, and result in a product of higher quality. A components that have been tested in other contexts than the one it was developed for, will have a higher quality. Several techniques within artificial intelligence has been applied, i.e., artificial neural nets (ANN), the structure mapping engine, and genetic algorithms.

We look at the reuse potential during the analysis phase of a software development project. Identifying reuse opportunities during this stage will have the potential to increase reuse during later stages of the process as both software architecture and implementation details may be reused. This is in line with e.g., Sutcliffe (2000) and others that work on reuse of requirements and domain analysis.

Artifacts produced during software analysis and design are typically requirements and models of various types. The types of models depend on the software development methodology used. To be able to reuse such models, their most generic properties should be identified. This will make it easier to find the best reuse candidates among which the developer can then choose.

According to Maiden (1992), evidence has been established that experienced software developers use analogies when solving new problems, while novices are mostly concerned with the programming constructs of the programming language itself. We suggest the incorporation of *analogical reasoning*, AR, as an integral part of the software development tools. The software developer may be aided through the process of creating software artifacts by getting suggestions of potentially reusable artifacts from previous projects. Thus, it is a goal to identify the best analogies for analysis models. The motivation is that analogies may turn out to be reusable, and that part of its solution may be reused in the new project.

Midttun (1998) implemented a prototype repository under my supervision, with an retrieval algorithm based on simple role name similarity during retrieval. This is discussed further in chapter 6. This algorithm could, however, not give a good semantic similarity measure. This model is replaced by the semantic similarity model described in this thesis, where role names can be picked from the whole vocabulary of an native English speaking person. The mapping phase is covered in addition to retrieval, and a genetic algorithm (Tessem, 1998b) is used during the mapping phase to identify good mappings. This new implementation is described later in this thesis. The repository contains far more models, and a large set of experiments have been executed. This set includes experiments that vary the amount of information that is stored about the reusable artifacts, i.e., *OOram* role models.

1.2.1 Choice of Analysis Models

OOram (Object-oriented role analysis modelling) models Reenskaug, Wold, and Lehne (1996) have been applied in this study. Appendix A gives an overview of OOram in general, with particular emphasis on the analysis phase. ROSA supports OOram role models in a simple form. Each role model consists of a set of roles, with one role being the *stimulus role* that initiates the activity in the model. A role can be connected to, or *knows about*, one or several other roles. A role may be able to send any one of a given set of messages to the other roles that it is aware of. An example role model is given in figure 1.1. Here the role *borrower*, marked with a dashed line, is the stimulus role. The small circles represents ports indication whether the role knows about a single (simple circle) or several (double circle) occurrences of the role at the other end of the path.

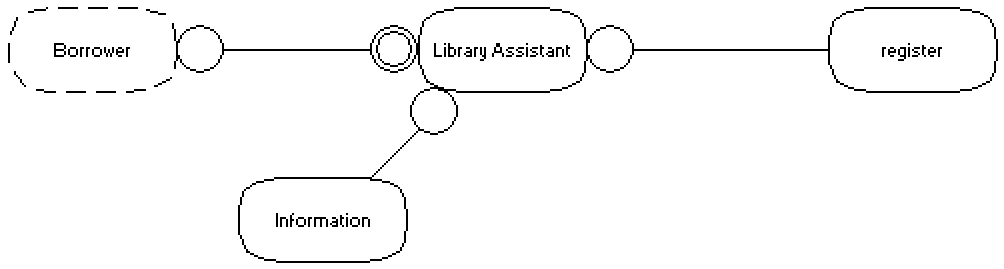


Figure 1.1: The OOram role model Request library card

OOram role models are chosen as examples of reusable components for two reasons. First, their intended use is to describe components during the analysis phase. Second, only the *roles* objects play are described, thus permitting the developer to delay considerations of what concrete classes will be used to fulfill a particular responsibility until the design phase. Activities performed during analysis are supposed to be directed towards the application domain, and it is an advantage to use a method that enforces this focus. Role models describe roles that communicate with each other, so it is not primarily a static description. They do not, however, show the message sequence as in *Unified Modelling Language* (UML) sequence diagrams. The diagram in UML 2 that resembles the OOram role models the most, are communication diagrams like sequence diagrams. These may, among other things, be used for modelling usage scenarios. A usage scenario is a description of a potential way a system is used (Ambler, 2004). The UML sequence diagrams does not, however, capture cardinality information on communication among the classes like the OOram models do for roles.

For reuse of software development artifacts to be possible, a large repository of such artifacts must be available. The repository should support traceability among components, implying that if a potentially reusable component is identified during analysis, it should be possible to study relevant artifacts from later phases to see if there is a real option for reuse. The ultimate goal is to reuse as much as possible of the relevant artifacts from later phases as well.

To be able to study this approach, techniques for knowledge representation, analogical reasoning techniques for retrieval and matching of analogical cases, as well as general techniques used in software development tools, including repository organization, browsing, and user interfaces, must be considered.

The choice of object-oriented modelling technique is not important, and the AR techniques proposed can most likely be used with any major object-oriented methodology. OOram was selected due to its focus on roles rather than classes. Today I might have chosen UML as the supported modelling technique, due to its much larger user community.

1.2.2 Use of Analogical Reasoning Techniques

Various techniques from AI, and particularly AR, have been tested within the ROSA project to evaluate their applicability for reuse of object-oriented specifications. Theory from AR, e.g., Gentner (1983), advocates that structural similarities is used to identify analogies. This theory is used as the foundation for the current approach. However, a methodology for using AR in software reuse must be fine-tuned with respect to what information is actually present in the repository of reusable components and what part of this information that is most likely to identify analogies that can be reused (section 3.1 discusses AR in more detail).

Thagard (1988) proposes a more pragmatic approach to analogical reasoning. He suggests that one should take into consideration how well the computational account of analogy corresponds to the experimental facts of human thinking, i.e., to identify what is the goal of using analogies. This implies trying to identify what is the purpose of using analogical reasoning and to try to use it in a way that fulfils that goal. I suggest a hybrid approach where a similarity measure is based on similarity of both structural and semantic information. How these similarity measures are to be balanced will thus be a question of analysis.

When a software developer performs analysis for a new project, he should have the option of using AR at any point in the process to automatically identify components from previous projects similar enough to be reused, in part or in whole. The system should give an estimate of the closeness of the match, so as to indicate the amount of work needed for modifications. The developer should be given access to all artifacts from both design and implementation of the project of choice so that he can realistically make his selection.

AR consists of 4-5 phases, where the first two are typically retrieval and mapping. Much research in AR focus on the mapping phase, where a potential reuse candidate is selected, and the system analyze whether it is actually a good analogy. However, for reuse to be successful, both retrieval and mapping should be given automated tool support. I propose an approach that, during the mapping phase, needs to analyze as few models as possible, and still ensure that the best possible matches are included in the selected set of models. This requires that the retrieval phase, although it uses a different and more imprecise approach for identifying reuse candidates than the mapping phase, will rank the models such that the best analogies are included among the top ranked models.

If potentially reusable artifacts are identified early in the process, I assume that artifacts from later phases may also be reusable. As I do not perform experiments where expert users identify the best analogies for a given role model, I assume that the best analogies are the ones that get the best mapping similarity. This assumption should be tested. In the following I briefly describe my approach of handling semantics and structure.

1.2.3 Semantics

To support analogical retrieval based on semantic information in role models, the repository has been extended with lexical content modelled after WordNet (Fellbaum, 1998c). WordNet contains all words of a person with English as his first language with these terms interconnected through various relations. This makes it possible to identify synonyms, specializations, as well as generalizations. All the roles in the models are named using terms from WordNet. If a phrase selected as role name does not exist (typical special collocations often used in models of various kinds), ROSA has been implemented to determine the semantic similarity without inserting additional terms, while still using WordNet relations to identify similarities.

1.2.4 Retrieval of Analogous Candidates

Much research within AR has focused on the mapping of properties from one situation or problem to another. One assumes that candidate analogies have already been identified. The mapping is but one phase of the process, however, while retrieval of potential candidates for analogy is another that must be carried out prior to mapping. In research within AR, the retrieval phase has been given less emphasis. Using AR to help identify reusable software artifacts, requires sharp focus on both these phases. Output from the retrieval phase, in the form of candidate models, is used to identify the best analogies during mapping.

Retrieval of potentially analogous models should be fast and result in the best candidates being identified. The best match should be among the highest ranked models because if the user has to inspect too many models, AR may not be worthwhile doing. To avoid an exhaustive analogical mapping of all role models, each of which can be arbitrarily complex, structural information about the models is calculated before a model is stored in the repository after its creation. My approach to retrieval based on structure and semantics is described in section 5.2.2. Since analogies are based on structural relationships, this information is searched to find structural similarity during retrieval. Experiments performed by Tessem (1998a)on synthetic OOram models indicate that retrieval should be based on a combination of a structural and semantic similarity.

1.2.5 Mapping of Base and Target Models

In the context of the ROSA project, several methodologies have been tried out for use during mapping, e.g., Ul-Haq (1997) tested the structure-mapping engine, SME, originally described by Falkenhainer (1988); Falkenhainer et al. (1989), Ellingsen (1997a) studied the use of artificial neural networks, and Tessem (1998b) tested the use of genetic algorithms. All these experiments were performed as stand-alone test applications with synthetic data. The technique that seemed most promising among these and that is selected for my project, is based on genetic algorithms. The implementation uses GAlib (Wall, 1996). Section 5.2.3 describes how genetic algorithms are used to optimize the mapping between two models based on both structure and semantics.

1.2.6 Scenario for Use of the ROSA Tool Prototype

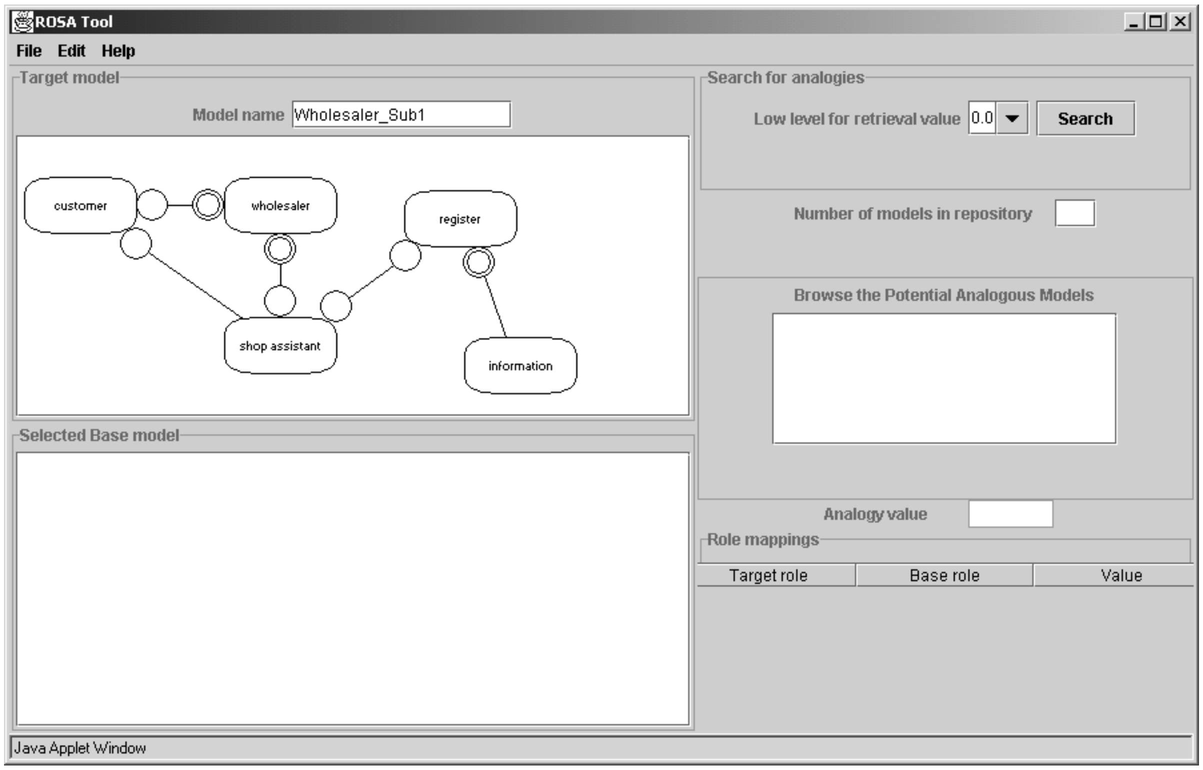
To illustrate how a software developer might utilize analogical reasoning, a prototype of the ROSA Tool has been developed. It shows how an AR facility can be used to identify analogies to a role model, and browse through the resulting set of models to evaluate the suggested models. Being a prototype, the tool does not include an editor for OOram models, although a model editor should be part of a ROSA Tool for it to be useful for a developer. Instead, all the OOram models used in the experiments were created using the Taskon OOram tool. These models were saved as gif-images with a name unique to each model. Thus the OOram role model editor can be simulated in combination with the AR facility both during search and when the evaluation of the results after the AR mapping has been performed.

Figure 1.2 shows the ROSA Tool after the user has selected an OOram target model. The user writes the name of the model for which he wants to see analogies[1]. If a model with that name exists, it is displayed in a target model window frame.

The user may modify one parameter for the analogy machine, i.e., a lower threshold for the probability of a base model being analogous to the target model after retrieval with a value between 0 and 1. A value of 0 means that all models in the repository will be included in the resulting list of analogies no matter how bad the mapping similarity is, and 1 implies that only identical models are retrieved. A systems developer does not necessarily have knowledge about AR, and the system has a default value of 0.6. If left unchanged, only base models with a probability of being analogous above 0.6 will be transferred to the mapping phase. In the example shown, however, the value has been changed to 0.0. All models in the repository are thus mapped to the target model.

When the analogy mapping is completed, the model names are displayed in the middle, right frame (see figure 1.3). The models are ranked according to their mapping similarity. The

[1]In a fully developed system a graphical editor would be used to draw the model.



Model name: Wholesaler_Sub1

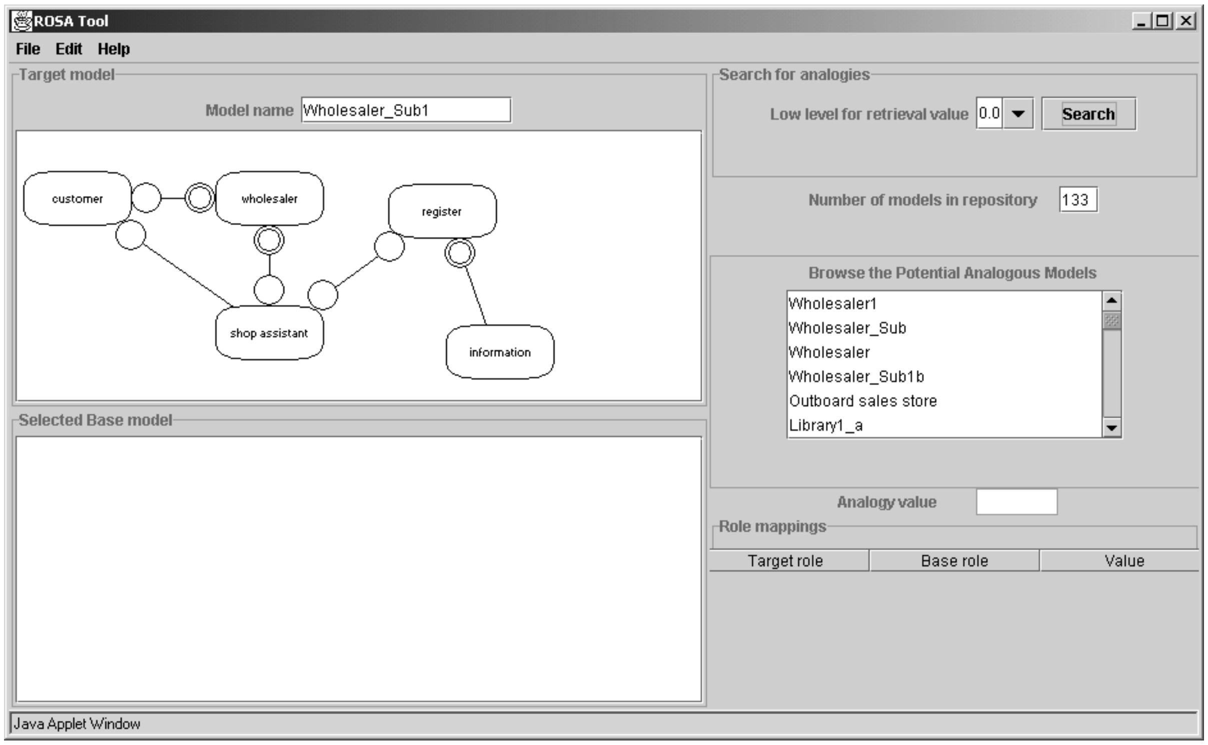
Low level for retrieval value: 0.0

Figure 1.2: The ROSA Tool with the target model displayed

number of models present in the repository is displayed, and in this case there are 133 models.

The user can now select a model or browse through the list. In figure 1.4 the user has selected a model that is displayed in the lower, left frame. The mapping similarity and the selected mappings between roles in the two models are displayed in the lower, right frame.

A support tool should let the user analyze further the properties of the suggested model to evaluate what parts of the previous project can be reused. If the tool had the functionality of the Tascon OOram tool, the scenario description, all attributes, methods, their parameters possibly with types, can thus be inspected. If the chosen base model has been synthesized, i.e., combined with other models in the project it belongs to, these models may in turn be inspected, and also, further design and implementation information may be analyzed to see how good the match is. However, these activities belong to later phases of AR, e.g., transfer. This, however, lies outside the scope of the ROSA project, and is not supported by the current prototype.



Model name: Wholesaler_Sub1

Low level for retrieval value: 0.0

Number of models in repository: 133

Wholesaler1
Wholesaler_Sub
Wholesaler
Wholesaler_Sub1b
Outboard sales store
Library1_a

Target role	Base role	Value

Figure 1.3: The ROSA Tool after the search has been performed

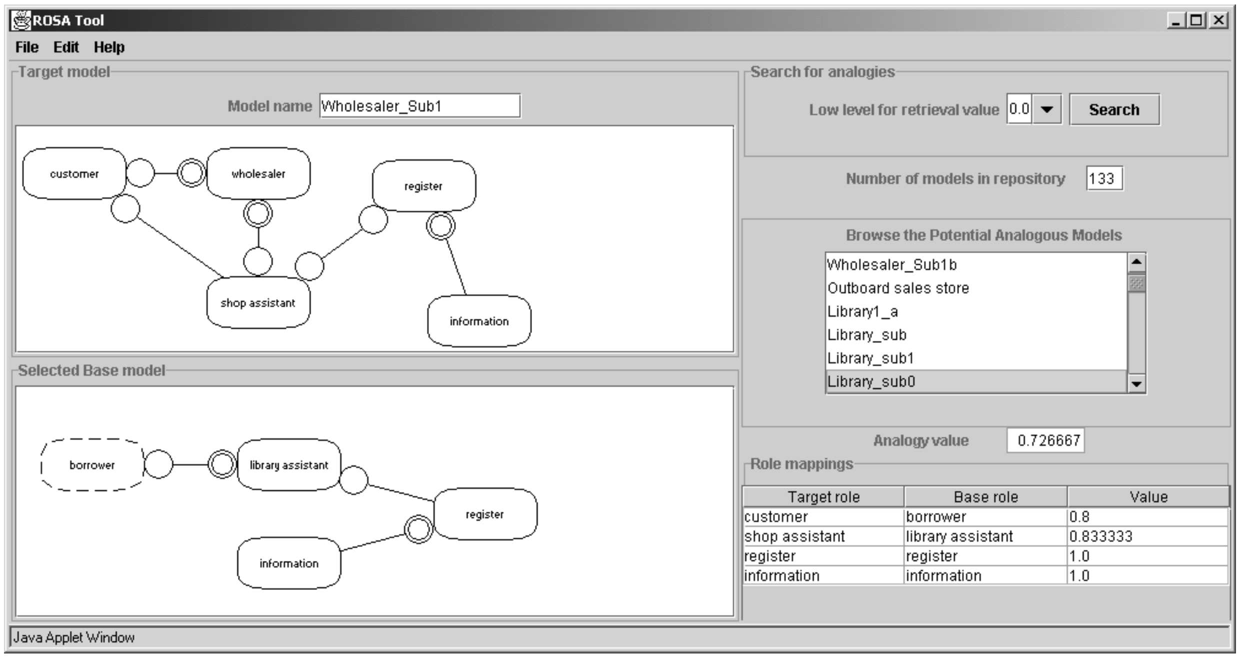
1.2.7 Discussion

This section presented an overview of the ROSA project. Previous experiments in the project have all been concerned with isolated aspects of the system, i.e., part of the repository or a technique used to identify analogies, using synthetic data, while my project intends to bring these aspects together.

In this thesis, OOram role models are used as examples of analysis artifacts. I believe, however, that the described techniques can be applied to any type of model that can be converted into a directed graph, and where the nodes in this graph have semantic information related to them. In the following chapters the problems that are the focus of this work, as well as the approaches chosen to solve them, will be discussed.

1.3 Problem Description

As mentioned above, I am interested in what properties of the analysis models that are important for identifying reuse candidates with as little effort as possible. I want to study if AR can



Model name: Wholesaler_Sub1

Low level for retrieval value: 0.0

Number of models in repository: 133

Wholesaler_Sub1b
Outboard sales store
Library1_a
Library_sub
Library_sub1
Library_sub0

Analogy value: 0.726667

Role mappings

Target role	Base role	Value
customer	borrower	0.8
shop assistant	library assistant	0.833333
register	register	1.0
information	information	1.0

Figure 1.4: The ROSA Tool after selection of a potential analogy

be used to identify reusable software development artifacts from early software development phases. Also, to be able to identify the candidates in an efficient way, I investigate how a repository of such components should be organized and how similarity can be measured during the retrieval and mapping phases of artificial AR.

The overall research question is stated as follows:

MAIN RESEARCH QUESTION: *Is analogical reasoning a viable approach to identifying potentially reusable artifacts from early phases of the software development life cycle?*

If potentially reusable artifacts, e.g. analysis models, are identified early in the process, artifacts from later phases may also be reusable. If so, reuse of analysis models paves the way for more extensive reuse in later in the process. The artifacts in question in this thesis are OOram role models. Due to lack of experiments involving expert users examining models to determine whether they are analogous or not, the thesis investigates the retrieval phase of artificial AR in particular, assuming that models with a high rank after analogy mapping are good analogies. The aim is to perform analogical retrieval and mapping where good analogies also get a good rank after retrieval. The above question is therefore operationalized to make it better suited for experimentation:

OPERATIONALIZED RESEARCH QUESTION: *Is analogical reasoning a viable approach to retrieving potentially reusable OOram role models from a test bed of models?*

The experiments performed address this operationalized research question. To make it even more concrete three sub-questions that can either be analyzed conceptually or tested experimentally, are stated. These are formulated in such a way that their combined answers address the above research question.

Q 1 *What kind of information is relevant to identify analogies between role models?*

Q 2 *How can we use this information to realize both retrieval and mapping of analogies?*

Q 3 *How can we balance this information in the identification of analogies?*

1.4 Research Design

Based on the research questions given above, I here outline the steps taken to define analysis and experiments. The main research question as given above is so general that it can, at best, be answered indirectly. The operationalized research question is more directly focused on the set of experiments that are preformed. To be even more direct, the experiments are directed towards the set of sub-questions that I have outlined.

To be able to answer these more detailed questions, analysis and experiments have been performed in a given sequence. They are ordered so that approaches and algorithms can be selected and parameters tuned before the final tests are performed. Answers to sub-questions may then be incorporated into the overall solution in order to give better overall results and a potentially more precise answer to the overall question. The following list outlines how the research questions will be addressed.

(Q 1) According to analogical reasoning, structural similarities are more important than surface similarities, i.e., names, for identifying analogies. However, Thagard (1988) claims that one should look at structural, semantic and pragmatic approaches. The repository must contain information about the structure of the models as well as a means to identify semantic similarities. In addition to the models' structure, the repository should contain information about the semantics of the models. The use of WordNet as a model for structuring the semantics of a term space for the repository is discussed in section 4.3.

In section 4.3.3, I discuss how a systems developer can pick a name for an OOram artifact, even if the name does not exist in the repository. I assume that all single, ordinary words exist in the repository. The problem may occur when a compound phrase, or collocation, is required. In section 5.3.5, I discuss what this implies for the semantic similarity model, and section 6.2.2 covers the implications for the design of the repository. In chapter 7, the proposed solution is tested.

To find out if inclusion of multiplicity information on the model ports improves the results of the mapping phase, an experiment that compares two situations; one with and one without such information. Multiplicity information is not used during retrieval, so it may only have an effect on the ease with which the genetic algorithm identifies a better mapping, and thereby on the rank of the base models in terms of their mapping similarity values. These experiments are discussed in section 7.2.5.

(Q 2) A semantic similarity model capable of identifying similarities among models in a large set of role models must be identified. A requirement is that it should be efficient during both retrieval and mapping. This is discussed in chapter 5.

1. Two semantic similarity models are evaluated to see how they behave under different circumstances before one model is selected for the repository. Such experiments are discussed in section 5.3.4.
2. These similarity models are tested on a set of role models. Experiments using the two similarity models were set up to fine-tune important parameters for the algorithms used. These experiments are described in section 7.2.1.

The system must handle problems with polysemy (same word with different meanings) and synonymy (different words with same meaning) without putting too much strain on the software developer and without spending too much time developing or enhancing the semantic database. These problems are discussed in section 4.3. To find out whether the lexical database is sufficient to identify semantic similarities, controlled experiments when it comes to use of synonyms and words that are semantically closely related, are needed. Experiments of this type are described in section 7.2.

To find out whether structural information of role models and names of roles are enough to identify analogous role models, I perform a set of tests where 24 different target models are used. The target models have been selected so they vary both in terms of application domain and size, the smallest difference being that one single role has a different word meaning. For all these experiments, the repository contain the same set of models. When describing the specific experiments, I will comment on the differences that exist between the specific target models (see chapter 7).

To find out how many models should be mapped to be able to identify the model with the highest mapping similarity (which should be the best candidate for an analogy), I carry out a set of experiments with different target models to see what rank the model with highest mapping similarity has after retrieval. This requires that tests must keep all base models also for the mapping phase in order to get all models' mapping similarity. See experiments in sections 7.2.1 and 7.2.5, where I analyze at what position the best analogy for 24 cases are found after retrieval.

A target model can be mapped to a model that is a synthesized model containing the target model itself (or a model analogous to the target model). If an analogy is found in such a situation, the extra information that this model contains might prove reusable, so it is important that these models are also retrieved. On the other hand, due to the way semantic retrieval is performed, large models might get a high score even though the mapping similarity turns out to be small. There could exist different situations where

1. all roles of both target and base models are mapped,
2. all roles of the target model are mapped,
3. all roles of the base model are mapped, and
4. there are roles in both the target and base models that are not mapped.

Experiments are performed where different types of penalties are given in the above-mentioned situations, and the results are analyzed based on how *good* the results are. Such experiments are discussed in section 7.2.9.

(Q 3) The analogical reasoning process should be efficient with high recall and precision ratios. Efficiency can be increased by reducing the number of base models sent to the mapping phase, which is the most expensive part of the process. To achieve this, it would be useful to identify a retrieval similarity value below which there will be no good analogies. If this is difficult, the user should be able to set such a value based on experience. This value represents the default lower threshold.

More interesting than the precise mapping similarities are the ranking of the models. The models that are found to be the best analogs after mapping, should preferably also be given a high rank after retrieval. The user should be presented with a list of models ranked according to decreasing mapping similarity.

Tests performed by Tessem indicate an equal weighting between structural and semantic similarity, and I take this as a starting point. It is important, however, to point out that Tessem's tests were not performed on role models, and that semantic similarity in his case was defined as having *identical name*.

Since the structural positions of roles in a role model are not taken into account during retrieval, the semantic distance between any two role meanings may have less importance during this phase. The reason is that the role in the base model with the highest semantic similarity is chosen without consideration for its structural position. Another point is that this phase should be as quick as possible, and there should not be added more complexity to the algorithms than necessary.

To decide whether structure and semantic similarity should be given equal weight, experiments using varying weights are set up. Such experiments are done on a set of target models. Description of such experiments is given in sections 7.2.7 and 7.2.8.

The results are presented in the following chapters. My approach to organizing the semantic information is described in section 4.3, the similarity model is discussed in chapter 5, and the ROSA prototype in chapter 6. In chapter 7, I discuss the environment and preparations for the experiments, the experiments themselves and their findings.

1.5 Overview of the Thesis

The rest of the thesis is organized as follows.

- Chapter 2 describes research within different fields of software reuse.
- Chapter 3 presents AI approaches, such as analogical reasoning (AR), case based reasoning and genetic algorithms, and discusses how these techniques have been applied for reuse of software components, particularly from within analysis and design.

The results are then presented in the remaining chapters through analysis and a description of the experiments that have been performed, together with suggestions as to where to go from here.

- Chapter 4 presents work on organizing semantic knowledge using the lexical database WordNet and describes my approach to integrate linguistic information in the description of OOram components. The intention is to see if this structure can be used in software development, and possibly during reuse of such components. The chosen approach is then compared to related work.
- A similarity model for the ROSA project is presented in chapter 5. This model will be used to find analogies, and experiments using this model may reveal that modifications are needed. The chapter also presents my approach to combine the structural and semantic information in the retrieval and mapping phases of AR.
- The new, extended version of the ROSA's prototype, and some aspects of design and implementation, are presented in chapter 6.
- Chapter 7 presents the experiments that have been conducted and their results.
- Finally, conclusions and ideas for further work are given in chapter 8.

The thesis includes the following appendixes

- A presentation of the OOram methodology is presented in appendix A.
- Appendix B describes the WordNet lexical database that is used to model the semantic information of the role models.
- The DTD for OOram role models is shown in appendix C.
- Example role models in XML format using this DTD is given in appendix D.

- An explanation of the Prolog files from WordNet that are used in the experiments is presented in appendix E.
- Finally, a list of the role models that are used in the experiments is given in appendix F. For each model there is information about its number of roles and the structure descriptions that are stored for that particular model.

1.6 Contributions

The contributions of this thesis can be identified within the following areas.

- Analogical reasoning used in a practical attempt to identify analogical OOram models, where the approach covers both the retrieval and mapping phases of AR.
- A semantic similarity model developed to identify similarity between OOram models using WordNet to model the semantics. The overall similarity model is based on both structural and semantic similarity.
- Experiments in which the relationships between structure and semantics are analyzed to optimize search.

The use of WordNet is particularly important as it secures the availability of a comprehensive term space developed by expert lexicographers. If the role models are created in cooperation with domain experts one can to a certain degree rely on the fact that terms from the domain have been applied in the models. If not exactly the same term has been used for a specific phenomenon in different models by different people, one would assume that the chosen terms will have at least some semantic similarity.

In this research experiments with several semantic similarity models during retrieval, have been undertaken. It turns out that a fairly simple similarity model based on WordNet gives a sufficiently good result. The more complex algorithms studied have given significantly better results. This indicates a pragmatic approach of choosing a simpler similarity model and using the saved time to map more models instead is fruitful.

Chapter 2

Software Reuse

The idea of software reuse can be traced back to McIllroy (1976) who says he "would like to see the study of software components become a dignified branch of software engineering [1]." Moreover, he would like to see "standard catalogues of routines classified by precision, robustness, time-space requirements and binding time of parameters." This idea was later picked up by Cox (1987), who envisioned that the use of object-oriented technology would allow us to buy software IC's off the shelf much the same way we buy hardware IC's. While this vision has not yet been attained, a lot of effort has been put into developing components, techniques, and tools to allow us to move in that direction. The closest we have come to this is the use of commercial off-the-shelf (COTS) software, that are built to be assembled into applications.

Deutch (1989) has defined software reuse as the utilization of an existing software component in an environment or for a purpose other than that for which it was originally intended, usually by changing some aspect of it and/or by using it as a component together with other components. In this thesis the emphasis is put on components that are not what you would typically call software components such as classes or module, but rather on artifacts from analysis.

Software artifacts that have been reused within different contexts are likely to be more robust and have higher quality than artifacts used only once. Each time they are reused, they are tested within a slightly new context, and yet undiscovered errors may be detected. Thus

[1]The term component, as used here and in Biggerstaff and Richter (1989), means any kind of software artifact. This is different from what is meant in component-based development: "A software component is a physical packaging of executable software with a well-defined and published interface"(Kobryn, 2000).

the reuse intensity of software artifacts can be considered a quality indicator.

The discussion of software reuse may be split along three important dimensions.

- *technology*, i.e., what technologies are available to support reuse—including the *software artifacts* we reuse and the *techniques* we apply (section 2.2),
- the *process* we apply when we either develop new software or reuse existing artifacts, i.e., how are the techniques incorporated in the software process itself (section 2.3), and
- the *organization* we try to build to enable and support this process; incentives and a culture for reuse are required.

Attempts have been made to incorporate aspects of all these dimensions into a methodology with accompanying tools. Some early attempts of such incorporation are the REBOOT (Ribot, Bongard, and Villermain, 1994) and STARS (Software Technology Adaptable Reliable Systems) (Creps, Prieto-Diaz, Davis, Simos, Collins, and Wickman, 1993) projects. A very different approach is described by McCarey, Cinnéide, and Kushmerick (2005) for use within agile software development projects. They propose to use an agent to recommend components from a repository. As there is little documentation or other support material produced in agile projects, the Rascel (Recommender agent for agile reuse) tracks the usage histories of a group of developers, under the assumption that the users are likely to use again components that they have previously been using.

In this chapter, reuse is discussed along the first two dimensions listed above[2]. While the technical dimension is the main focus of this thesis, I also illustrate how the suggested techniques can be incorporated in a software development tool. The organizational dimension, however, is completely left out, although this omission does not imply that it is considered unimportant. First, however, I look at what has been described as the reuse problems.

2.1 The Problems of Reuse

Although the types of software artifacts that can be reused may vary, the problems we face when trying to reuse them are similar. Four problems related to reuse in general are identified by Biggerstaff and Richter (1989). They are to:

[2]A previous version of this chapter can be found in Bjørnestad (2001).

1. find components
2. understand components
3. modify components
4. compose components

Of these, they consider the *understanding* problem to be the most fundamental. It is important that the user acquires a mental model of the part to be reused, and assisting tools should therefore be offered.

One main problem when it comes to reuse of code components is that they must be general enough to be used in slightly different contexts, yet specific enough to apply effectively to the design tasks at hand. This problem has to do with *how* the components, e.g., classes, are designed. To find the components most suitable for solving a particular problem, we must study what are the best criteria to describe the components.

Gall, Jazayeri, and Klösh (1995) draw problems from the experience made with software reuse. One of these problems can be categorized within the technology domain, namely, what is the unit of reusability, i.e., what is the right component, and how to build it? Their answer to this question is mainly directed towards the software development process, and I shall therefore delay the discussion of it until then. They also emphasize, however, as have many others, that software has to be *designed for reuse*. For reuse to be successful, the techniques chosen must be fully integrated within the software development process.

Within the organizational dimension, several problems have been listed. For example, Zand et al. (1999) emphasize that for people to be willing to reuse software artifacts, the available artifacts must have a certain degree of credibility, i.e., how do we assess the reliability of a reuse approach? So far, not many experience reports have been published, and Tracz claims that if such projects fail, they are not likely to be reported. A study of the barriers, both personal and organizational, of adoption of a reuse program, is presented by Sherif and Vinze (2002). They present a series of propositions that try to explain what is the cause of such barriers.

Of the problems listed above, I will focus on the first two, namely finding and understanding components. As mentioned elsewhere, however, the software process must be taken seriously, and it should be made feasible to integrate the techniques that are developed into a software development tool.

2.2 The technical dimension of reuse

This dimension can be seen in terms of the software artifacts we reuse, and the technologies we use to achieve software reuse. The two main categories of approaches to reuse will guide which types of components will be reused: 1) Constructive approaches that focus on structuring new artifacts, or applications, from existing components. 2) Generative approaches using a high-level specification language to automatically generate an application, either by inference rules, transformational rules, or other types of formalizations Biggerstaff and Richter (1989). The discussion in this thesis, however, will focus entirely on the constructive approaches. These approaches have also received the most attention thus far.

Throughout the last decades, software reuse has become more and more emphasized, and a shift can be seen from a predominant focus on reuse of code towards reuse of artifacts from earlier phases of the software development cycle. The types of reusable software artifacts reported in the literature range from code fragments and simple, well-defined functions in the early days, through the use of class libraries, to the current situation where there is no limit to what can be reused, at least in principle.

Efforts reported concerning reuse of code are discussed in section 2.2.1. Thereafter, reuse of software artifacts from analysis and design is discussed in section 2.2.2, and the techniques applied to enhance reuse are discussed in section 2.2.3.

2.2.1 Reuse of code

During the early history of software reuse, the main focus was reuse of *code*. Third-generation languages had built-in constructs for writing functions or operations as abstractions over basic program statements. Libraries of simple functions were built, particularly within domains such as mathematics and graphics, where a clearer understanding of the domain existed. Here, each function has one clear, logical meaning in the application domain, and it is easy to separate it out in one short, simple function with a simple interface. The programmer was thus freed from low-level details of coding and could concentrate on an increasingly higher level of abstraction. At the same time it did not require a great investment of time to understand the functions. In many other domains there is not always such a clear understanding of how to split the functionality into separate units or components, and there is often a much greater variability in required interface to other components.

The code produced and the problems solved have become increasingly more complex over the years. At the same time we have seen a move towards increased use of data encapsulation or information hiding. There is a development via *abstract data types*, *object-oriented classes* and *class libraries*, to *frameworks* (Johnson and Foote, 1988), the use of design patterns (Gamma, Helm, Johnson, and Vlissides, 1995), and *software components* (Brown and Wallnau, 1998; Kobryn, 2000). The sequence of this listing goes towards more and more complex artifacts where increasingly more design knowledge is captured. The key point is to make software artifacts as self-contained as possible in order to ease their combination with other components. However, this is not as simple as was originally envisioned. Classes that are not designed to work together will probably not do so, except in the simplest cases (Berlin, 1990). The last three of these technologies will be discussed in the next section, as I consider them to be more reuse of design.

The more complex a component is, the more design information has been made part of it. By this I mean information about how the component is designed to work together with other components i.e., what role it plays in an application. According to Johnson and Foote (1988), frameworks are examples of such design. Whitehurst (1995) refers to reuse of frameworks as systemic reuse. I discuss frameworks in the next section.

Object orientation, more than any other programming approach, supports data abstraction, encapsulation, and information hiding. Reuse is made possible through the definition and modification of classes. Modifications can safely be made because the information belonging to an object, the object's knowledge, is encapsulated within this object rather than distributed throughout the system, and the side effects of changes are thus minimized. A class is an atomic entity that describes the common set of behaviour among a set of objects. Objects are instantiated from existing classes, and new classes are created, through *subclassing*, as specializations of existing classes.

Programming in Java, Smalltalk, or other languages where rich class libraries are available, requires the programmer to have a good understanding of the class library. The user must know where to look for candidate classes for reuse, and what the responsibilities of the individual classes are. The Smalltalk-80 class hierarchy contains hundreds of classes and thousands of methods. Accordning to Rosson and Carrol (1990) this can be quite confusing for a programmer not familiar with Smalltalk, and it may take several years before a programmer becomes fluent in using the library. The Java library[3] contains 2098 classes organized in 135 packages of differing size and complexity as well as 625 interfaces. The number of methods is close to 20,000, although use of polymorphism makes the number of unique method names much lower. To give an indication of this, the library contains 267 versions of the `toString()` method. For each new version, the complexity increases as new packages, classes and methods are added for solving new problems. Another complicating issue is that when some methods and even classes are replaced by others, they are not removed from the library, but merely marked as deprecated. One could say that the bigger the reuse potential, the bigger the problem of understanding. In the cases where an application framework is available, the user must understand the design philosophy behind the framework, or the *design and architectural patterns*, if used, instead of every single class in the system.

To overcome the finding problem in agile development, McCarey, Cinnéide, and Kushmerick (2005) propose the use of a recommender system. The motivation is twofold: 1) Recommend software components that the developer is interested in. 2) Recommend useful components that the developer may not be aware of. The components that they refer to are Java methods. An recommender agent tracks usage stories of a group of developers. Recommendations are made using a collaborative filtering approach with use of content-based filtering to order the recommendations.

2.2.2 Reuse of Software Artifacts from Analysis and Design

The motivation for identifying potential reuse prior to the implementation phase is that when a design has been completed, available code components may be unsuitable. Small differences in design decisions may determine whether reuse of a specific class is possible or not. The alternatives that exist in the cases where incompatible design choices have been made, are

[3]Java 2 platform, standard edition version 1.4.2

either to re-design or to code from scratch. Tracz (1994a) stresses that it is becoming more and more apparent that one needs to reuse more than just code to gain increased benefits from reuse. In order to achieve an order of magnitude improvement in software productivity, Tracz (1995) claims that one must reuse products from the software development life cycle other than just code.

Application Frameworks

Experience has shown that, although “an intellectually seductive idea” (Tracz, 1994b), it is more difficult than first envisioned to combine components from different vendors. Such problems can be caused by differences in abstraction mechanisms, granularity, and component interfaces which make the components incompatible. The idea of an application framework was introduced as a mechanism to enhance reuse. Johnson and Foote (1988) define a *framework is a set of abstract classes that are designed to act together as a skeleton for an application or part of an application*. Frameworks support reuse at a higher level of granularity, and reuse of a framework is really reuse of architecture and design.

Each abstract class in a framework solves a specific part of the overall task at hand. For each application, the abstract classes of the framework must be replaced with concrete realizations of the abstract class. A framework can be seen as a main architectural type of construct to support reuse.

A framework is typically designed to solve a particular set of problems, e.g., graphical user interfaces (GUI) or simulators for a group of related problems. One example of a GUI framework is the Model-View-Controller framework (MVC) (Krasner and Pope, 1987) built into the Smalltalk-80 system. Due to its success, the MVC is now widely used within GUI programming. Frameworks are believed to have a greater potential for reuse than general-purpose class hierarchies. However, although successful within specific domains, e.g., GUI, frameworks do not seem, so far, to have been as successful as anticipated.

Frameworks are typically described as being either *horizontal*, solving one type of problem across application domains, for example, representing the GUI or distributed communication, or *vertical*, built for applications within a domain, e.g., health care systems Booch (1994). The most widely used frameworks today belong to the first group. Frameworks for more narrow domains have also been proposed, for example for the financial domain (Birrer and Eggenschwiler, 1993). Research has been conducted both in the organization of such frameworks and in methodologies for creating frameworks. An example of this is found in Riehle (1999). In his work, Riehle discusses the problems of designing, using, and learning frameworks.

Design Patterns

Because software reuse has been less successful than many people had hoped for, new approaches have been taken to reach the goals to get more reliable and cost-effective software systems. One of the most predominant of these is *design patterns* (Gamma et al., 1995) or, rather, pattern *languages* Berczuk (1994). A design pattern can be defined as a solution to a recurrent problem within a context. Fowler, however, defines a pattern as "an idea that has been useful in one practical context and will probably be useful in others" Fowler (1997). He uses such a loose definition to stay close to the underlying motivation of patterns, without adding too many restrictive amendments. An important point is that successful solutions are described in a form so that they can be reused.

The application of patterns within software engineering was inspired by an architect, Christopher Alexander, who had the notion that ordinary people could design their own homes by the use of architectural patterns that where written to solve particular problems within a greater context of building houses Alexander et al. (1977). He wrote his patterns as parts of a pattern language. In the early 1990s Alexander's ideas were picked up in the object-oriented community, and they were presented at OOPSLA workshops on software architecture.

The solutions the patterns prescribe have typically been developed and evolved over time, and will not be the first solutions we choose for new problem we encounter. Such evolution has typically happened to increase reuse and flexibility in the software. A rule of thumb says that the solution should have been used in at least three different contexts for us to call it a pattern. This leads to another point that needs to be stressed, and that is that patterns are identified through practice. They cannot be invented. An idea stems from a project, and if is proven through practice that is useful in other settings as well, it can be formulated as a pattern. Schmidt (1995) claims that

> patterns aid the development of reusable components and frameworks by expressing the structure and collaboration of participants in a software architecture at a higher level than source code or object-oriented design models that focus on individual objects and classes.

A pattern should have four essential parts: a statement of the *context* where the pattern is useful, the *problem* that the pattern addresses, the *forces* that play in forming a solution, and the *solution* that resolves those forces. It is important that a pattern has a name, as this enriches the vocabulary of development, and when developers that are familiar with patterns use these names, there will be less sources of misunderstanding among them.

While the patterns published by Gamma et al. (1995), where relatively low-level design patterns, it has since then emerged a great variety of patterns and attempts to write pattern languages. A patterns handbook was published by Rising (1998). This was an attempt to collect all the, at that time, known design patterns published elsewhere by others. Her motivation was for these patterns to be more widely known and used. Since then patterns have emerged within more specific areas such as enterprize application architectures Fowler (2003). The previously mentioned MVC framework has been described in patterns form Buschmann, Meunier, Rohnert, Sommerlad, and Stal (1996), and these patterns are now also built into the Java API. Patterns have become popular and people have described patterns within other areas than design, e.g., analysis (see below). This has led to using the term software patterns to cover all these different types of patterns.

Analysis Patterns

As a natural consequence of the maturing field of design patterns, work on analysis patterns was presented by Fowler (1997). He describes patterns from conceptual business models that provide key abstractions from domains such as trading, measurement, accounting, and organizational relationships. Fowler stresses that such patterns are based on conceptual modelling that are linked to interfaces (types) and not to implementations (classes). In addition, Fowler also describes what he calls support patterns which show how analysis patterns fit into an information systems architecture. Simple analysis models are more likely to result in solutions that are easy to maintain and reuse.

Another important point that Fowler (1997) makes is that it is not always clear when domains are the same or different. He uses the example of health care models that he found appropriate to describe financial analysis within a manufacturing company. Problems of diagnosis and treatment could be used to understand the causes of high-level financial indicators. He suspects that there are a few highly generic process models that can be used across traditional boundaries of system development and business engineering. This, he says, raises some serious questions about the promised development of vertical class libraries for industry sector, and that perhaps the solution is that the true business frameworks will be organized along abstract conceptual processes instead of along traditional business lines.

Problem Frames

While a design pattern looks inwards towards the solution of a problem, Jackson (2001) has defined a *problem frame* as something that looks outwards towards the world where the problem has been identified. A problem frame thus "defines the shape of a problem by capturing the characteristics and interconnections of the parts of the world it is concerned with, and the concerns and difficulties that are likely to arise". Like a design pattern a problem frame describes a recurring situation, and like a design pattern language it provides a taxonomy that can be used during the analysis. If a part of a problem is identified with a recognized problem frame, experience associated with that problem frame can be taken advantage of.

Problem frames are intended to be used during analysis. They may help you to structure and analyze your problems as a collection of interacting *subproblems*. Although different real-world problems are very different and need different solutions, they may contain subproblems that are similar and that may have similar solutions. These sub-problems can be classified into classes of problems, that may be used to solve similar problems later on. These simple subproblems are captured in problem frames, and they appear, according to Jackson (2005) as ingredients or aspects of realistic problems. Each problem frame constitutes what we might call a problem pattern. Jackson resembles these with analysis patterns (Fowler, 1997).

Components

Another approach is the development and use of *software components* (Brown and Wallnau, 1998; Herzum and Sims, 2000). A component, according to component-based development, may be defined as *a unit of code that implements one or more clearly defined services as defined by a set of published interfaces*. A component can be defined as having the following properties (Szyperski et al., 2002): 1) is a unit of independent deployment; 2) is a unit of third-party composition; 3) has no (externally) observable state. Herzum and Sims (2000) add the requirement that a component has a well-defined, clean interface or set of interfaces that are accessible at run-time. This implies that a regular Java class is not a component. Also, they claim that a component must be of a medium-sized to very large granularity, i.e., a JavaBean or ActiveX control is *not* a component.

Herzum and Sims also require that a component can be easily combined and composed with other components to provide useful functionality. Only through collaboration can an individual component achieve its usefulness. Further, they list four essential characteristics of a software component model: the component, the component socket (where the component is plugged in), the ability of components to cooperate with other components, and the user of the component.

The last point emphasizes that some components are intended to be used by designers, while other components are intended for end-users, and knowledge about who the user is will tell you what skills to expect from the user. To a developer, an example component might be one that manage addresses. Such a component is called an *enterprize distributed component*. To an end-user like a business analyst, on the other hand, a payroll system might be considered a component. A component like a payroll system is called a *system-level component*.

In recent years, the idea of components has over-shadowed much of the interest in object-oriented libraries and frameworks. Components have been seen as a way to implement Cox's vision of software IC's (Cox, 1987). One of the success stories of components is the use of Visual Basic components. Fowler (1997) states that the reason for this success is that all the components are based on one common framework—the Visual Basic environment. Within other types of information systems such a common environment may not exist. Other types of software components include Enterprise Java Beans, and Microsoft's COM and .NET technologies. Components may include libraries, objects (such as COM or CORBA components), class frameworks, etc. The growth of a component industry is seen, for example, within the Java community.

Popular IDE's can be populated with components which the developer can use to build his application. Components can be plugged into application frameworks that depend on the components features and not on their state. If components are built using design patterns to help developers to understand how they can and should be used, there is greater possibility of success. The terms component-based development and service-oriented architecture has, according to Tracz (2004) to some extent replaced reuse of software.

In the call for papers to The First International Workshop on Engineering Component Based Systems (ECBS04), it is stated that "Component-based Software Engineering (CBSE) is concerned with the development of software intensive systems from reusable parts (components), the development of such reusable parts, and with the maintenance and improvement of systems by means of component replacement and customization." This is an indication of an increased maturity in the field of reuse.

Domain Analysis

One of the reportedly most successful areas of software reuse is domain analysis (Tracz, 1994b). If a domain analysis has been performed within a specific application domain, it can be assumed that this domain is better understood, and the knowledge gained can be reused in applications developed within that domain. Neighbors (1989) has defined domain analysis as "the activity of identifying the objects and operations of a class of similar systems in a particular problem domain". When studying a domain, commonalities within the domain, rather than knowledge specific to one application, are in focus (Lung and Urban, 1995a). The analysis performed can therefore be used to build other applications within the same problem domain. Typically, the approaches suggested for domain analysis share the purposes to 1) manage identification, 2) capture and evolve domain knowledge, and 3) make this information reusable for creating new systems within the same application domain.

Domain analysis can be viewed as an expansion of conventional requirements analysis. The main advantage it provides is flexibility (Lung et al., 2007). It is time consuming, and it is typically suited for domains that are well understood. Arango (1994) describes domain analysis methods. According to him, an object-oriented view considers domain analysis to be modelling of the real-world actors, objects, and operations, as opposed to program and data structures that will be the subjects of object-oriented design.

Effort towards construction and reuse of requirement specifications was initiated by the ESPRIT Nature project. According to Maiden and Sutcliffe (1994) domain knowledge is built into tools in the form of *domain abstractions*. These abstractions represent the fundamental behaviour, structure, and functions of a domain class. Domain knowledge describes generic types of systems as well as specific applications (Maiden and Sutcliffe, 1992). Sutcliffe (2000) discusses this domain knowledge theory and how it can be used to generate reusable generic models. For such models to be reusable there should be design rationale connected to them.

Lung and Urban (1995a) discuss the classification of domain models that can be used to find analogies between domain models, particularly when trying to model a domain that is not so well understood. They use a classification scheme that combines hierarchical and faceted classification (see section on Classification on page 36).

Another way to summarize the knowledge captured within domain modelling is described by Fowler (1997) in terms of analysis patterns. This was discussed in a previous section.

Poulin (1999) even claims that the problem of domain analysis has been solved. He also states that, without domain analysis, an organization cannot hope for more than about 20 % reuse of the total amount of software development.

More recently, Frakes (2002) uses the term *domain engineering* to denote the process of “analyzing a domain, recording important information about the domain in domain models, creating reusable assets, and using the domain models and assets to create new systems in the domain. It has two phases: domain analysis and domain implementation.”. One could say that this description includes both the understanding and the attempt to turn the captured understanding into reusable assets.

Requirements

When analyzing a problem and extracting requirements for a new system, several iterations may be necessary before the system is complete. This is particularly true if the system is within a domain that is unknown to the analysis team, or if the users of the system have little experience in formulating their requirements for the future system. If reusable requirements could be identified, these could be reviewed, and the users or the analysis team might discover shortcomings in their original specification which could be adjusted accordingly. If all, or parts of, the requirements are very similar, it might be that (part of) the design and implementation could also be reused.

Lam, Jones, and Britton (1998) suggest the use of an operational reuse model to help understanding of the process for managing requirements reuse. They also say that such reuse is most valuable within specific domains and that work on domain modelling is important.

Heumesser and Houdek (2003) report on work at DaimlerCrysler where they recycle requirements from one car model to the next. In this way not only the specification fragments can be reused, but also the test specifications that were written for the original one as well as the underlying experience. Requirements are split into categories based on whether they are dependent or independent on a particular car model.

For each functionality they describe all the requirements in the model-independent part. While doing this, they use only abstract interfaces and parameters to avoid the dependent information. The model-dependent information is supplied through the use of wrappers, which offers instantiation information for all the interfaces and parameters that were identified.

Heumesser and Houdek (2003) describe three different levels of reusability of requirements: 1) Domain requirements that can be considered stable in a given environment, 2) Functionality requirements that deal with functionality of the system under development, and 3) model-specific requirements that are specific for a given car model. Requirements at the first level can be reused as black-box components. Requirements at the second level are similar from model to model, but not identical. The MVC design pattern is used to split the user interface aspects from the processing aspects, and to separate out what part of the processing requirements that are part of the general functionality that are reusable, and those that are part of the model-specific ones. The model-specific requirements, on the other hand, are seen as less reusable.

2.2.3 Techniques Applied to Software Reuse

Research and development relevant to software reuse has been conducted within a diverse set of areas, including how we build reusable artifacts, and what techniques and other means can be used within software development environments. A technique can, according to Basili (1989), in this context be defined as a basic technology for constructing or assessing software, e.g., reading or testing.

Language constructs

Efforts have been made to develop languages with constructs important to reuse, e.g., *information hiding*, *inheritance*, *polymorphism*, and *interfaces*. In Java, if a programmer writes implementations towards an interface instead of a class, he is free to substitute one class for another as long as the classes implement the same interface. Meyer (1987) has introduced language constructs to enhance reuse and reliability, such as pre- and postconditions built into the code itself, in Eiffel. Meyer (1992) claims that these constructs make the language more suitable as a design language.

Use of language constructs to enhance reuse seems to be somewhat restricted, though. If these were the only types of techniques to be used, we would be restricted to reusing software *code*. So, although it is not irrelevant, it is not sufficient. In this thesis I do not discuss reuse based on such techniques.

Software developed for distributed programming needs some standard way of communication. Components need a protocol described through an interface. IDLs (Interface Definition Languages), are developed for CORBA (Common Object Request Broker Architecture), for example, and WSDL (Web Services Definition Languages) are defined for web services. The separate construct called an *interface* in Java specifies what messages an object implementing that interface will understand. The implementation of these methods is the responsibility of the classes that implement the *interface*. This is an important feature that underlies today's component-based development (Kobryn, 2000).

Whitehurst (1995) discusses a set of language-level constructs that support reuse. These are relational extensions, called “associations”, which are designed to allow the relationships found in design patterns, frameworks, and abstract architectures to be represented explicitly.

Design of class hierarchies or libraries

A trend towards *open source* and public domain code has given people access to large class libraries, e.g., Java Swing and C++ STL libraries. The widespread use of such libraries has increased testing and thus acceptance of these libraries. Commercial libraries also exist, and these libraries are heavily used. When such libraries have wider applicability, the belief increases that the investment will pay off.

A class library is not static. It will necessarily change during its lifetime. Such changes will not be only additions. There must be an ongoing attempt to monitor the library development to prevent the relationships between the components to break, or to be too complex after modifications have been made. Problems have been reported when a class hierarchy gets too deep. A deep hierarchy increases the understanding problem. This has been the case particularly with the Smalltalk library as all classes are organized in one hierarchy.

A technique called *refactoring* has been suggested by Fowler (1999). The idea is that whenever a modification to existing code is necessary, simplifications should be initiated before or during the required modifications. This technique has advice for how and when modifications should be performed to both method and class structure. Existing IDE’s, such as IntelliJ, NetBeans, and Eclipse, all support a subset of the refactorings that Fowler suggested.

It has been pointed out that, to avoid subclasses that only need part of their super-classes’ behavior, subclassing should always be made as pure specialization. Otherwise, classes are much more difficult to reuse due to their size and complexity. The *understanding problem* also increases, as it is more difficult to see what part of the definition is actually required. Such problems emphasize that standards are used when classes are designed. A design tool that could aid in the reorganization of the class hierarchy based on special rules of what was a *good* subclassing practice—following the standards—would be useful.

Software libraries are often organized according to a hierarchical scheme, e.g., a class hierarchy. They usually use single inheritance. When using languages that allow multiple inheritance, the classes do not form such a hierarchical structure, and this may lead to several types of problems. For example, if an attribute or method is inherited from two different ancestors, a decision must be made as to what occurrence to use in the subclass. When defining relationships among components, it is often not clear what is the correct classification, as we can think about several dimensions of the collections of components.

Repository organization

A software repository is a database that is designed to store the types of items that software developers need when they are developing new applications. The artifacts can be of any kind, depending on the repository at hand, such as objects and components, patterns and frameworks, test cases, and information about the same. Such information can amount to when they were created, goals and functionality, where they are used, and so on.

Historically, software artifacts, i.e., code, have been kept in a regular file system, and it has not been of high priority to maintain and organize the self-developed code. A repository may contain projects with all the components that are developed during the project's lifetime. A repository should not be part of any project in particular, as its content should be something created of things developed in the various projects. Such a framework should be an integral part of a CASE tool.

Retrieval

Suggestions to use techniques from *information retrieval* (IR) Baeza-Yates and Ribeiro-Neto (1999) for enhancing reuse have been put forward. Other approaches have suggested the use of intelligent browsing to identify reusable components (Pintado and Tsichritzis, 1988). A system might accept a description of an artifact with a certain set of properties or facets (see section on Classification below). Based on this description, the repository is searched for the best match. Such a search might either result in an artifact that completely matches the description, or in one or more artifacts with a partial match. The user should then be able to select the artifact that resembles the requirement the most, and perform any needed action to adjust the artifact to better suit the current needs.

When designing an application, it would be useful to have a tool that, based on a description of its components, would search the repository for components that match the description.

The search should preferably start while the description is being made, so that the system developer avoids unnecessary work. This would be a more or less intelligent browser or retrieval tool. Techniques from AI may be used to retrieve reusable components (see section on Techniques from AI on the current page).

Classification

Above, I discussed the design of class hierarchies, and the problems that relate to use of multiple inheritance. From a reuse perspective, it can be useful to be able to describe components along different dimensions.

Faceted classification has been suggested as a means to describe reusable artifacts so they can be more easily identified. It was originally proposed by Vickery (1960) as a technique to organize library information, to help people who may be interested in searching for different types of information about books and journals, for example author, title, topic, or genre. Prieto-Díaz (1985) suggests that it be used for software components in order to ease reuse, and it has been applied in several projects, e.g., the REBOOT project (Sørumgård, Sindre, and Stokke, 1992, 1993) for classification of software components. Lung and Urban use faceted classification to classify domain models (Lung and Urban, 1995a,b). However, Tracz (1994b) lists faceted classification as one of the areas where reuse has not been very successful.

Techniques from AI

The AI techniques used for reuse have a wide range, for example

- knowledge representation and cognitive graphs used to represent knowledge (Borgida et al., 1985),
- retrieval based on heuristics, maybe combined with indexes (Lee and Harandi, 1993),
- artificial neural nets (ANN) used for identification of similarity between a problem at hand and previously developed examples (Merkl et al., 1994),
- genetic algorithms used to find components with specific properties (Tessem, 1998b),
- case based reasoning (CBR) to identify previous cases that are similar to the current case (Fouqué and Matwin, 1993; Maguire et al., 1995), and

- analogical reasoning (AR) to identify reusable *base problems* based on a current *target problem* (Lung and Urban, 1993; Chaiyasut, Shanks, and Swatman, 1995; Maiden and Sutcliffe, 1996; Spanoudakis and Constantopoulos, 1996a),
- content-based and collaborative filtering used in a recommender system (McCarey, Cinnéide, and Kushmerick, 2005).

AR has been used to identify reusable problems or artifacts across domains, while in CBR the goal is to identify similar cases within the same domain. In chapter 3 I present in more detail the use of analogical reasoning (section 3.2), case-based reasoning (section 3.4), and genetic algorithms (section 3.6) within the field of software reuse.

2.3 The Process Dimension of Reuse

Software reuse is still mostly associated with reuse of code. As analysis and design are more general, artifacts from these stages are likely to turn out subject to reuse more frequently. This requires that reuse is integrated in the software development process itself, and that all types of software artifacts from previous projects are available from a CASE tool repository. It also requires that the software development process used be adjusted to take advantage of reusable assets of different types.

Existing IDEs like IntelliJ IDEA from JetBrains, Eclipse, NetBeans, SUN^{TM}ONE Studio, Smalltalk-80, and Eclipse contain a browser—a software tool for searching through a collection of existing categories of object classes, search utilities, and a repository. In Smalltalk-80 the classes are organized in categories, while in Java they are organized in packages. All the environments have several ways to help the user search and reuse classes, as well as identifying classes and methods.

IBM has integrated Rational Rose^{TM}with the IBM software, e.g., IBM WebSphere. Users can create analysis and design models in Rational Rose^{TM}, export code to WebSphere, and then reverse engineer the code. Such a solution may permit models from analysis and design to be stored in the same, or adjacent, database as the classes and packages. This could permit the search options to be made on components from the analysis and design phases of a project, and thus permit reuse on a much wider scale. A problem may occur, however, when the user does not really know what to search for. Since tools such as those mentioned above started to emerge, the amount of available code in the program libraries has exploded, but to learn the structure of the libraries has not become any easier.

Although IDEs are becoming better tools to develop software, they are still a long way from some of the visions that have been put forward. A CASE tool should support design *with* and *for* reuse (see e.g. the REBOOT project (Fagerhus and Karlsson, 1992)). Basili (1989) describes an *experience factory* containing all types of knowledge about the process and components that relate to the the software projects. When looked upon as a component factory, the following aspects are important.

An overview, according to Basili, of how an experience factory may be used, is given below:

- During requirements definition the analyst negotiates with the customer, uses the factory to see what already exists, and negotiates the requirements based upon this knowledge. The more component information is added to the factory, the more knowledge is made available for reuse.

- During specification and design, the requirements will be turned into a system design and specification for the required components. Components that can be well specified, can be turned over to the experience factory, and orders will be filled for components.

- During integration and evaluation, the task is to integrate the components into its own specified design. The components may be returned to the factory for later reuse.

This is a simple description of the process. It does not really explain how the reusable components are actually identified. It is assumed that the analyst knows the component factory. It is important that the selected techniques to search for and to identify reusable components are able to do so precisely and effectively. A hidden problem exists, however. If the factory has been developed by several people with different backgrounds, it is likely that they have used different terminology and that their focuses have been different. When the analyst no longer knows the content of the factory, he will have problems finding what he is looking for, or rather, he will have problems formulating his queries in a way that will be successful. This is one of the problems addressed by this thesis.

Tracz (1994a) argues: “Integrated CASE tools, when they get here, have the potential to enhance software reuse. … They will facilitate the traceability of requirements to design and code, along with the other artifacts associated with software development.” It should be possible to know which components from any phase in the development cycle contribute to fulfilling which requirement. Previous reports of unsuccessful reuse of requirements, see e.g. Lewis (1990), may have been a result of tools that are not good enough. CASE tools should incorporate support for maintenance, testing, domain-specific modeling, project management, and quality assurance (Forte and Norman, 1992).

Maintenance of the reuse repository is also a problem. For a repository to be, and remain, useful, we cannot merely add components to it. Components must be documented better if they are expected to be used in later projects, and they may have to be rewritten to be more generic to enhance their reuse potential. Information about how the components have previously been used, may be useful in situations where they are considered for reuse. The addition of new artifacts may require modifications of relationships among existing ones as well. Traceability of change should be supported, as this reduces the problems of maintenance. It also makes the system’s components more reusable over its lifetime (Parnas, 1994).

While Herzum and Sims (2000) state that we have not reached the stage where we can build applications by assembling, adapting, and wiring together existing components into a variety of configurations simply because the components does not exist yet, Wallnau, Hissam, and Seacord (2002) claim that today, software development is to a great extent done by integrating components developed by others into a framework. I will not argue one way or the other, but only stress that for reuse of components to take place, the needed components must be available. To the extent that the components are available, this is another important argument for integrating reuse in the software development process.

A component-based software engineering (CBSE) process, which relies on reuse, has emerged. This relies on a large base of reusable software components and some integrating framework for these components Sommerville (2004). A CBSE process, according to Sommerville (2004), should consist of the following steps:

- Outline systems requirement—the user requirements are done in outline rather than in detail, and the stakeholders are encouraged to be as flexible as possible when defining their requirements.

- Identify candidate components—search for suitable components, select and validate the appropriate ones. This activity is unique to CBSE. Components may be found in a local database or from trusted suppliers.
- Modify requirements according to discovered components—this is done in cooperation with the stakeholders if suitable components are not found. Users may be willing to slightly modify their requirements if it results in cheaper software or earlier delivery.
- Architectural design—during this stage you decide on a component model if it has not already been done during component search. The high-level architecture is determined, and decisions about distribution and control are made.
- Identify candidate components—reiterate over the component identification as some of the components previously chosen may turn out to be unsuitable. This may in turn lead to new requirement changes, so the process is in nature iterative.
- Compose components to create system—if the components have not been written for the current application, adaptors must often be written to reconcile the different component interfaces.

2.4 Conclusion

This chapter has tried to give an overview of the reuse problems, along with some suggested solutions. Of the problems related to reuse that are listed, finding and understanding components are the most difficult ones. In this discussion, I have covered a rather wide area, primarily to indicate the diversity of both the types of software artifacts that can be reused as well as the various techniques that have been suggested used to increase reuse.

There has been a move towards the reuse of artifacts from earlier phases in software development. At the same time the available code in libraries has moved from function, through modules, classes, frameworks, to components in the sense that Cox (1987) envisioned then. They have thus become more complex and important design knowledge are built into them, e.g., they may play specific roles in a framework or design pattern.

My focus is reuse of artifacts from analysis and design, and in particular the reuse of analysis models. If tools and techniques exist to support such reuse, the reuse potential could be recognized early on in the development phase, and the lower level components used in previous projects will also be available for reuse.

I have discussed reuse of requirement specifications, domain analysis, problem frames and analysis patterns in this chapter. These are all examples of assets from the analysis phase that may be reused. As a matter of fact, many of these techniques were proposed for the purpose of making available knowledge that will help in the development of future systems. However, while these techniques help in the understanding of the domain and possibly also the components that are a result of the analysis as well as the sharing of this understanding, they do not help with the problem of finding the right artifact to reuse.

The identification of reusable components is a main problem in software reuse, and if techniques and tools were available that could help the developer do this, this would make a great impact. If a component-based approach for software development is used, the user requirements may be described in analysis models, and a search for similar models may be performed. The identification of such models may in turn lead to the reuse of software components.

In the next chapter I discuss the main focus of this project, namely the use of analogical reasoning, case-based reasoning, and genetic algorithms to help solve problems related to reuse of software artifacts.

Chapter 3

Intelligent Approaches to Software Reuse

Section 2.2.3 gave an overview of techniques from AI that have been applied for reuse or software assets. In this chapter some of these AI techniques that are relevant for my approach to software reuse are discussed, and an outline of how they have previously been applied in relation to software reuse is given. In section 3.1, problem solving through analogical reasoning (AR) and approaches to computerize such attempts are discussed. Next, in section 3.2, examples of how AR has been used within the area of reuse of artifacts from early stages of software development are presented.

The use of case-based reasoning (CBR) in software reuse is discussed in section 3.4. The main difference between CBR and AR is that while AR tries to identify similarities across application domains, CBR does the same within an application domain. The reasons for including a discussion of CBR in this chapter are that it is similar to AR in goal, although the similarity search is restricted to one domain, and that within CBR much work has been put into the retrieval phase, and this work may turn out to be useful also within AR.

Genetic algorithms (GA) are used to find optimal solutions when the search space is huge and when an exhaustive search would take an unrealistically amount of time. Section 3.5 discusses genetic algorithms in general, and in section 3.6 I present an approach to apply GA for the analogical reasoning in software reuse. Finally, section 3.7 discusses how the theory presented in this chapter influences my research.

3.1 Analogical Reasoning

Analogical problem solving is important for how humans solve problems. Transfer of concepts and relations from one domain to another lies at the bottom of both common-sense reasoning, learning, and complex problem solving. Contributions to the field of AR stem from research directions such as psychology, cognitive science and AI. Because AR has been used in so many different contexts and manners, AR is also a problem-solving strategy that is difficult to fully understand and formalize. Some theories that model specialized aspects of AR have emerged. General and complete theories, if at all possible to develop, are, however, yet to be seen. In this section I give an overview of some theories.

3.1.1 Analogical problem solving

Analogical problem solving can briefly be described as an attempt to reuse and adapt a problem solution in a new situation, maybe within a completely different domain. Similar properties between the two situations are identified, and knowledge that is not mapped may then be transferred from the first (base) situation to the new (target) situation. For a more comprehensive overview of early work within the field of computational approaches to AR, see (Hall, 1989). In the following I describe attempts to computerize analogy and to give a computational model for the concept.

Early use of analogy in computer systems includes work by Evans (1968) on geometric intelligence test problems, and Kling (1971) on theorem proving. Carbonell (1983a) introduces the term *transformational analogy*. This is based on the idea that one can transform the solution of a previous design problem to a new one by a limited set of transformation rules. In *derivational analogy* Carbonell (1983b) adds knowledge about the decision steps performed in the production of the previous solution. The purpose is to help to choose the transformation steps in the search for a solution to the new problem. Winston (1980, 1982) describes a system that learns new principles based on a precedent story and an exercise that matches this precedent. The exercise describes a general situation, and the precedent describes a specialization. The two are matched, and a feature of the precedent is reformulated as a more general principle in the terms of the exercise. Later work has focused on more general approaches, and structure mapping is discussed in section 3.1.2, pragmatic approaches in section 3.1.3, and case-based reasoning in section 3.3.

The steps that are involved in computational models for analogical problem solving vary among the authors, partly due to different focus of the work and detail. Kedar-Cabelli (1988) describes the 5 following steps:

1. *Retrieval*—from a target case, search the knowledge base for a *base* case that resembles the target and has a reusable solution.
2. *Mapping*—match attributes, relations, causal chains from the base case to the target case.
3. *Justification*—ensure that the derived properties, obtained from matching, are in fact valid. If not, correct or modify.
4. *Transfer*—consider if any unmapped parts of the source may be reused in the target.
5. *Learning*—accept useful computed properties of the target and add those to the knowledge base (Evans, 1968), derive general knowledge from the base and target solutions (Carbonell, 1983a; Winston, 1982), or improve the target solution by using additional analogies (Burstein, 1988).

The most difficult of the five steps described above seems to be *retrieval* of base cases. The process should ensure that the retrieved cases are relevant and can be mapped to the target case. Many systems developed so far lack support for this step, and they expect the user to supply an analog base case. The most promising methods seem to involve indexing by common generalizations, negative associations (reasons for wrong classification), prototypes and important differences between cases. Such techniques are described by Kolodner et al. (1985), Simpson Jr (1985), and further improved by Bareiss, Porter, and Wier (1988); Porter, Bareiss, and Holte (1990). These ideas will be elaborated in section 3.3 on *case-based reasoning*.

Search for common abstractions is the most common approach to the mapping step. Important contributions include work by Gentner (1983) on the systematicity principle, and work on goal-oriented mapping, by e.g., Thagard (1988). These contributions are discussed in sections 3.1.2 and 3.1.3. Justification will typically be a manual task, as it is the human user that must decide whether the analogy is valid or not.

3.1.2 Structural mapping and systematicity

Gentner (1983) introduces the concept of structural mapping and the principle of *systematicity*. Structure mapping theory focuses on the mapping phase and not on retrieval of a potential analog case. Several systems have been built using algorithms based on this theory, e.g., the Structure Mapping Engine by (Falkenhainer et al., 1989). Gentner addresses the problem of *concept learning by analogy*. The atom is used as an example. How can we learn more about the atom, given the analogy "The atom is like our solar system"? By using this question, we assume that previous knowledge about the solar system is present.

The basic intuition of structure mapping of analogies is that a system of relations among objects or concepts, known to hold in one problem domain—the *base domain*—also holds in some other domain—the *target domain*. Gentner's work has shown that people, when trying to find analogies to a given problem, tend to map interrelationships among the facts rather than surface similarities. This implies that objects from the base problem do not necessarily have to resemble objects from the target problem. For example, given the analogy mentioned above, what parts of the atom are mapped as analogous to the sun? According to Gentner's theory, the sun's surface attributes such as *hot*, *yellow*, and *massive* will not be mapped. Higher-order causal relations such as "Sun attracts planet" and "Planet orbits sun" are more likely to be mapped to identical relations in the target, i.e., "Nucleus attracts electron" and "Electron orbits nucleus". As can be seen from this example, the nucleus of the atom is mapped as analogous to the sun.

The method for finding analogies described above is often termed the *systematicity principle*. It states that we should prefer to map causal networks of relations over independent and unrelated relations. By preferring causal networks, we constrain the number of possible mappings between base and target for any analogy.

Although recognized as important, Gentner's work has received a fair amount of criticism (Holland, Holyoak, Nisbett, and Thagard, 1986; Kedar-Cabelli, 1988), e.g., that the structure mapping theory assumes that all causal relations describing the base are candidates for mapping. It is certainly difficult to distinguish the relevant causal relations from the irrelevant.

3.1.3 Pragmatic approaches

Thagard (1988) presents two criteria for evaluating computational approaches to analogy. First, how well does a computational account of analogy correspond to the experimental facts of human thinking? Second, how powerful is the computational approach: What tasks does it perform, and how efficiently does it perform them?

According to Thagard, computational approaches to analogy are often organized along four general dimensions: representation, retrieval, exploitation, and learning. Note that this model is somewhat different from the one described by Kedar-Cabelli (1988) presented above. Along each of these dimensions we can distinguish between syntactic, semantic and pragmatic approaches. Syntax deals with the properties that symbols have due to their form. Semantics is concerned with the relations of symbols to the world, and finally *pragmatics* deals with the use of symbols. Once a representation is constructed, it can be stored in memory, but in order to be used in an analogy, it must first be *retrieved*. There are several approaches to retrieval: Either providing an analogue explicitly, indexing by common features, goals and failures, problem-directed spreading activation, or undirected spreading activation.

Depending on problem complexity, one or several of these approaches can prove to be fruitful. Thagard concludes that indexing and *spreading activation*, described in (Holland et al., 1986), provide the most plausible kinds of mechanisms for finding potentially relevant analogues. Ideally, these kinds of retrieval should be combined.

When a potential analogue has been retrieved, it must somehow be exploited. There is much discussion on how the mapping process should be performed. Thagard presents some alternatives: Syntactic comparison using relational structures (structure mapping (Gentner, 1983)), loose pragmatic comparison using elements of problem solutions, tight pragmatic comparison using pre-represented ingredients, or integrated syntactic, semantic, and pragmatic comparison using multiple constraints (Thagard, 1988).

The pragmatic approaches pay more attention to the goals of the user of the analogy when attempting to determine what parts of the analogy to use. An analogy should only be accepted if it supports problem solving. An integration of syntactic, semantic and pragmatic techniques will make powerful cross-domain analogies possible.

3.2 Analogical Reuse During Analysis and Design

During the 1990s, there was an increased interest in the use of AR for reuse of software artifacts, and particularly within reuse of specifications. In this thesis, I focus on attempts to use AR, and to some extent CBR, within reuse of software artifacts from early phases of software development, i.e., analysis and design. This includes, among others,

- Maiden (1992), Maiden and Sutcliffe (1992, 1993, 1994, 1996) — analogical specification reuse during requirements analysis and analogical reuse of domain analysis.
- Lung and Urban (1993, 1995a,b) — combine classification of domain models with the requirements set by AR to aid the reuse of domain models.
- Spanoudakis and Constantopoulos (1992, 1996b) — present a computational model to find similarity between specifications for AR.
- Bellinzona, Fugini, and Pernici (1994) — reuse of specifications.
- Cunningham, Finn, and Slattery (1993) — knowledge engineering requirements in derivational analogy.
- Lee (1992) — bottom-up, case-based approach to domain analysis.
- Lee and Harandi (1993) — retrieval for software design reuse, and Harandi (1993) — the role of analogy in software reuse.
- Whitehurst (1995)— the understanding and application of AR in relation to software development process; systematic software reuse in the domain of system architecture or framework, i.e., design reuse. Language extensions to make reuse easier.
- Jeng and Cheng (1993, 1994, 1995) — a formal approach to reuse more general components and specification mapping.
- Fernandez (1998) — describes building systems using analysis patterns by finding analogies between the applications.
- Chiang and Neubart (1999) — constructing reusable specifications.

Some of these approaches will be categorized and discussed below.

3.2.1 Reuse of Artifacts during Analysis

This section covers attempts to reuse specifications, requirements, domain models, and analysis patterns. The terms may be used slightly different by different authors, but I will try to clarify this whenever needed.

The Advisor for Intelligent Reuse (AIR)

Maiden and Sutcliffe (1994) use AR in a tool called a *domain matcher*, that is a hybrid computational mechanism for AR between a requirement specification and one or more domain abstractions. They use techniques similar to SME (Falkenhainer et al., 1989). The domain matcher integrates structure mapping algorithms and heuristic-based reasoning with domain semantics to infer consistent fact-pair and object-pair mappings with abstractions. It consists of four parts: a match controller, structure matcher, rule-based matcher, and domain discriminator (Maiden and Sutcliffe, 1994). When a requirement specification has been matched to an object system model, the result is a quantitative measure of similarity and two sets of local mappings describing object-pairs and fact-pairs with object system.

The domain matcher is supported by an intelligent *dialogue manager* which explains retrieved domain abstractions to requirements engineers. A *problem classifier* uses mappings inferred by the domain matcher to detect incompleteness, over-specialization, inconsistencies, and noise in the requirement specification. The problem classifier is integrated with a cooperative tool for explaining complex problem situations to requirements engineers.

Classifying Domain Models in Support of Analogical Reuse

Lung and Urban (1993) integrate domain analysis in an analogical approach to software reuse. They stress that reuse cannot occur without domain analysis, and that the requirements of AR should be part of the *standard* products of the domain analysis process. Their analogy-based domain analysis methodology consists of the following steps:

1. Identify the requirements and features needed by AR.

2. Incorporate additional constraints, i.e., object behaviour and system dynamics, into the mapping to reveal a higher level of analogy to identify differences between *base* and

target systems and detect potential evolution of the problem domain.

3. Define steps and identify products of the domain analysis method to meet the requirements of analogy analysis. Networks of causal relations are derived from the diagrams produced in step 2. Constraints on causal relations adopted in this approach include goals, object behaviour, and system dynamics.

4. Perform domain analysis and generalize models into a reusable form for future development and analysis. A generalized model is used as an *interface* for mapping across domains.

5. Evaluate the process and products with domain experts. The domain will evolve and the model may be extended.

When the domain at hand is poorly understood, a model could be compared to a well-defined domain (Lung and Urban, 1995a). For this to be a viable approach, domain models must be classified. Lung and Urban present a hybrid classification scheme based on hierarchical and faceted classification, i.e., it consists of a hierarchy of facets, where each facet describes a key aspect of the software (see Classification on page 36). Different components could typically be classified into several sub-categories. A faceted classification scheme is thus more flexible than a hierarchical one when it comes to describing such a situation. It may be difficult, however, to decide what facets to choose for the components during domain analysis. They propose a hierarchy containing three layers, each representing a separate level of abstraction with a set of unique facets: 1) the application *domain*, 2) the main functions and relation types of the domain, and 3) the application.

Facets in layer one are used as classifiers, e.g., the *application domain* facet ensures that applications within the same domain are grouped and retrieved together during the analogy phase. Facets in layer two bridge the gap between the generic domain abstractions and features in specific application domains. The *function* facet is used to describe key functions of the system. Another facet in this layer, the *relation class*, depicts semantic relations between critical objects and is used during the analogous problem solving. Facets in layer three, general software attributes, are used to bridge the gap between the domain and a specific application or a set of design alternatives. Two analogous domains with different complexity levels may need different design approaches. In cases where a domain contains a set of sub-domains, an iterative process can be applied to examine and select appropriate facets.

Elaborating Analogies from Conceptual Models

(Spanoudakis and Constantopoulos, 1992) focus on the problem of “defining a quantitative similarity relation between descriptions of software artifacts so as to promote their analogical reuse”. They present a generic method for computing similarity between object-oriented qualitative descriptions. They emphasize efficiency and usability, and do not only model human cognition.

Their specifications are written in the Telos Language. Analogies are elaborated by functions that measure conceptual distances between objects with respect to semantic modelling abstractions. Their similarity model is discussed in section 5.1.2.

3.2.2 Reuse of Artifacts from Design

This section describes creation and reuse of artifacts from design.

Automated Acquisition and Refinement of Reusable Software Design Components

Lee (1992) presents a framework for a hybrid software design environment that supports reuse of design schemes or cases contained in a knowledge base. A design schema is an abstraction of a family of software designs for a particular application domain, while a design case represents the design of one single application.

The focus is on acquisition and refinement of design schemes, and this is implemented in a prototype called CAReT (Component Acquisition and Refinement Tool). It contains an analogy-based retriever that first attempts to retrieve design schemes that are most similar to the new design case. If no applicable schema is found, the cases are searched.

There are two main functions: initial design schema acquisition and subsequent design schema improvement. Both use a common underlying derivation mechanism. The correspondence of features is established using domain knowledge, if available. The features are examined to determine common patterns of operations and objects. Common features are then abstracted. If domain knowledge is not available, a similarity metric is used to compute the degree of closeness among features.

When design cases are searched, the retrieved cases are ranked, and chosen are those belonging to the same design family as the new design case. If the *case retention threshold* is exceeded, the new design case is used together with other cases from within the same design family to derive a schema; otherwise it is stored along with other cases in the same design family. If case retrieval fails or no case is deemed suitable, i.e., its design family is unknown to the system, then a domain dictionary is created for the new design case. This mechanism automates domain analysis by identifying the basic objects and operations for a domain-specific design schema, i.e., a domain model for applications within that design family.

Design schema derivation has six stages: 1) mapping and abstraction of objects and attributes, 2) mapping and abstraction of operations, 3) validation of design schema, 4) creation of refinement and specialization rules, 5) cataloguing of design schema rules, and 6) cataloguing of object and attribute mappings.

Systemic Reuse of System Architecture

Whitehurst (1995) uses AR to support software reuse of system architecture in the form of software frameworks. The approach is based on a general mechanism for building reusable software by refactoring object-oriented designs to remove inter-object dependencies. An inter-object coupling is removed by extracting it into an association with the role-specific behaviour. This keeps object class specifications small and focuses on intrinsic characteristics. At the same time, a mechanism is provided whereby objects may be composed into frameworks, which represent prototypical application architectures. Finally, this approach utilizes an augmented object-oriented class hierarchy as its knowledge base.

During system design, an automated assistant monitors the progress, and attempts to form analogies between the system under development and known system frameworks. If a framework is accepted, the user is provided with candidate objects with partially mapped concepts and role-based behaviour, based upon the analogical relationships. The presented objects are modified to fit new application requirements.

Whitehurst introduces language-level constructs for reuse support that are relational extensions, called *associations*, that let relationships found in design patterns, frameworks, and abstract architectures be represented explicitly. He argues that by refactoring class relationships into *roles*, the classes become more reusable.

The methodology uses a knowledge-based repository for creating, organizing, and managing reusable software artifacts. A modified Structure Mapping algorithm is used to recognize analogical relationships between partial systemic specifications and existing system frameworks. Whitehurst's similarity model is presented in section 5.1.3.

The reuse agent knows about applications based on the particular framework, so the use of the framework in those applications can serve as a prototype for the new application. A framework is the main architecture of the program, and the developer must supply individual objects compliant with the requirements of the framework to specialize it to a particular application domain. The approach taken in those applications to provide the framework with ancillary objects can also be reused in the new application.

3.3 Case-based reasoning

Case-based reasoning has been given several definitions. One definition is the use of solutions to old exemplar problems in the search for solutions to new problems within a specific domain. It differs from AR in that its reuse is domain specific. AR occurs in transfer of solutions from one domain to another. The boundaries between the two approaches are vague, however. One particular property of case-based reasoning is the domain-specific knowledge, which helps the system in search for candidate solutions to the problem at hand.

Theory from case-based reasoning may contribute to the retrieval phase of the analogy process. Most case-based systems use some kind of indexing to describe cases. This is considered the only efficient way to retrieve relevant cases, e.g., Kolodner (1983) or Ashley (1988)).

The PROTOS (Bareiss, Porter, and Wier, 1988) system supports search for relevant cases, and it also classifies cases into categories. It uses four kinds of indexes:

1. *positive reminder* — a relation from an attribute to a particular category or exemplar —

a case

2. *censor* — a reminder that weakens an association from a particular attribute to a category
3. *exemplar link* — a connection from an exemplar to its category
4. *difference link*—a link that explains important differences between categories

The system has the possibility to learn categories and their prototypes, as well as indices and their strengths (Porter, Bareiss, and Holte, 1990).

3.4 Case-based Reasoning and Software Reuse

In this section a few projects within CBR and software reuse will be describes. These are

- Ostertag, Hendler, Díaz, and Braun (1992) — retrieve reusable components described by (*feature, term*) pairs by measuring the similarity between the features of the components.
- Fouqué and Matwin (1993) — CAESAR, a case-based approach to software reuse.
- Adachi, Kobayashi, and Ohta (1994) — support for software design using case-based reasoning

These approaches will be discussed in the following.

3.4.1 AI-based Reuse System (AIRS)

Ostertag et al. (1992) describe the steps of reusing components as: 1) *define* a new component in the form of its functionality and its relation to the rest of the environment, 2) *retrieve* candidate components that are similar to the *target* under development, 3) *adapt* the best candidate, typically through a set of modification steps, and 4) *incorporate* the new component into the reuse library.

The classification model is based on features used to describe the components. Each feature is defined as a finite set of related values called *terms*. The set of features used to classify a collection of components within a certain domain defines a *feature space*. Each component is modelled by a finite set of pairs of a feature for a given feature space and a term of that feature. A component defines a mapping from features to terms. A feature that is not relevant for a component is mapped to a special null term.

Components are compared based on their descriptions, and AIRS quantifies their degree of similarity by computing a distance between their corresponding descriptions. AIRS provides two comparators, the *closeness* and *subsumption* relations, that are used in computing distance. Subsumption is used to represent the case in which a new component is expected to be directly realizable using a component currently in the library. This computation is based on the classical AI notion of property inheritance in semantic networks. The closeness relation is added to capture the notion that new components can be constructed via the modification of existing constructs. I describe these relations in more detail in section 5.1.1.

3.4.2 CAESAR

CAESAR (CAse-basEd SoftwAre Reuse) (Fouqué and Matwin, 1993) describes a case-based reasoning framework, where cases consist of program specifications and corresponding C language code. The case base is initially seeded by decomposing relevant programs into functional slices using algorithms from dataflow analysis.

CAESAR retrieves stored specifications from this base and specializes and/or generalizes them to match the user specification. Testing techniques are applied to the construct assembled by CAESAR through sequential composition to generate test data which exhibits the behaviour of the code. For efficiency, inductive logic programming techniques are used to capture combinations of functions that frequently occur together in specifications. Such combinations may be stored as new functional slices.

3.4.3 Software Design Support

Adachi, Kobayashi, and Ohta (1994) suggest a system based on CBR where the system designer gets help, from previous design cases, about how to perform the design of a new project. When a task has been finished, the system should help in deciding what task to perform next, or what item to construct next.

3.5 Genetic Algorithms

Genetic algorithms take their inspiration from natural genetics, where individuals can be identified by its combination of genes. In this section, I first give a brief introduction to natural genetics before going on to discuss computational genetics.

3.5.1 Natural Genetics

In the simplest of life forms, each individual consists of one single cell that reproduces by *fission*, i.e., each cell is split in two, and the DNA is copied into each offspring. This results in two cells that are identical to the original one. Evolution in such environments happen through *mutation*, i.e., genetic changes caused by, for example, environmental influence or damage, and where each off-spring becomes slightly different from the parent. Often these individuals are failures, and they cannot compete for food or they die out of other reasons, but sometimes the mutation is successful in the sense that they get a competitive advantage over other individuals in the environment.

In more advanced forms of life, with sexual reproduction, each individual has genes organized in *chromosomes*, and two individuals that produce off-spring, each provide one half of the chromosomes needed to make up a complete set for an individual. By combining chromosomes the off-spring will always be different from both the parents in some ways, and the variability among the individuals in a population will be much more varied than without sexual reproduction. Mutations also happen within such populations. Evolution is the sum of all semi-random events and natural pressure to reproduce or die.

3.5.2 Computational Genetics

Genetic algorithms have their background in work by John Holland in the 1970s (see for example Koza (1998)). He showed how an evolutionary process can be used to solve problems by means of a highly parallel technique that is now called the genetic algorithm (GA). Genetic algorithms are, according to Tessem (1998b), particularly well-suited for finding solutions to problems of discrete nature with many local optima.

A GA transforms a *population of individuals*, each with an associated *fitness value*, into a new generation of the population, using the Darwinian principle of “the survival of the fittest” and analogs of naturally occurring genetic operations such as *crossover* and *mutation* (Koza, 1998). Crossover implies sexual recombination of genes.

A GA used for problem solving, according to Luger (2002), typically consists of the following three distinct steps:

1. The individual potential solutions of the problem domain are encoded into representations that support the necessary variation and selection operations. These can be as simple as bit strings, but they can also be more complex representation forms, e.g., production rules.

2. Mating algorithms produce a new generation of individuals that recombine features of their parents. In addition, mutation algorithms are used to introduce random changes.

3. A *fitness* function judges which individuals are the best life forms. By this we mean: What are the individuals that are most appropriate for the eventual solution of the problem.

After the individuals are evaluated, they are compared against each other. In each generation, individuals who will produce off-spring for the next generation, are selected. The termination conditions can be based on different criteria, e.g., a maximum time limit, a solution’s value must be above a certain threshold, or the solution’s value does no longer improve between generations.

The preparatory steps required to use the conventional genetic algorithm on fixed-length character strings to solve a problem, are according to Koza (1998): 1) the representation scheme, 2) the fitness measure, 3) the parameters and variables for controlling the algorithm, and 4) a way of designating the result and a criterion for terminating a run.

Once the preparatory steps for the algorithm has been set up, the genetic algorithm, as described briefly above, can be run. The initial population is randomly generated. The evolutionary process is driven by the fitness measure. The primary parameters that are set to control the genetic algorithm are the population size, and the maximum number of generations to run (Luger, 2002). Additionally, there may be some secondary parameters to set. This set may vary based on the implementation of the genetic algorithm.

3.6 Genetic Algorithms and Reuse

Tessem (1998b) presents an application of genetic algorithms during the mapping phase in analogical reasoning. This work was done in the context of the ROSA project. Genetic algorithms are used to identify correspondence between entities and relations in the two situations, these situations representing OOram role models.

Tessem (1998b) models the analogical mapping problem as the problem of finding subgraph isomorphisms in two graphs describing two potentially analogous situations. Each situation is represented by a directed graph. The mapping function, f, is represented by a binary matrix M where each row represents a target node (t_i) and each column a base node (b_j). If the element $M_{i,j} = 1$, then $f(t_i) = b_j$. The rest of the elements in a row is 0.

The search is further constrained by considering the semantics of the entities and relations in the situations. The problem is then to optimize the weighted sum of semantic similarity between mapped nodes and the structural preservation in the mapping.

The fitness function used in Tessem (1998b) counts the number of *local isomorphisms* in the mapping, denoted $I(M)$, between two directed graphs, where a local isomorphism is a pair of edges, from each of the two graphs, $(t_i, t_j) \in E_T$, and $(b_k, b_l) \in E_B$, such that $f(t_i) = b_k$ and $f(t_j) = b_l$. E_T and E_B are the edges of the target and base graph respectively. The higher $I(M)$ gets, the higher the structural consistency in the mapping is.

In addition to taking account of the structural mapping, semantic similarity between the mapped nodes should also be considered. While Tessem says the semantic similarity should be computed between the nodes by some kind of spreading activation in a term space, in his work it is fixed before the actual mapping takes place. The strength of the semantic match $S(M)$ between the target and the base is given by

$$S(M) = \sum_{i,j} s(t_i, b_j) M_{i,j}$$

where $s(t_i, b_j)$ denotes the semantic similarity between the nodes t_i and b_j

The total fitness function is then given by

$$O(M) = \alpha S(M) + \beta I(M)$$

The experiments were run on synthetic data, where one OOram model was first generated. Additional models were created as variations of the first model. Random, semantically similarity values were created for all node pairs, with a tendency to have high similarity values for those who map in the ideal mapping, and lower for the rest of the pairs.

During experimentation, $\alpha = 1$ and β is set to a value between 0.25 and 2. While Tessem discusses whether to use pragmatics during the mapping, of the type used in ACME, i.e., to let the goal of the analogy problem constrain the search, he has, in this work, chosen not to do so. The types of pragmatics he mentions can be used is to punish the objective function with a large number if a node or a set of nodes is not mapped.

During experiments, Tessem tests four different genetic algorithms (Tessem, 1998b) and each of them used with four different values of β (0.25, 0.5, 1, 2). The performance of the algorithms were measured by comparing how many of the ideal mappings the algorithms were able to identify. The algorithm giving the best results uses ten populations, each with ten individuals. Each population runs a steady state algorithm (Syswerda, 1991), and for each population one individual migrates to another population each generation. This algorithm is called a *deme* genetic algorithm according to (Wall, 1996). Tessem argues that the reason for the success of this algorithm is that the problem demands high diversity to get good results, and that this is ensured by the multiple population algorithm. The deme algorithm with $\beta = 0.5$ gave the highest percentage of correct mappings.

3.7 Discussion

This chapter has discussed three research areas within AI that have been applied in relation to software reuse. These three approaches: AR, CBR and GA are in different ways relevant for the ROSA project. Each of these approaches are described, both theoretical and how it has been applied to research within software reuse. The examples of their use within software reuse are selected because they all focus on reuse of artifacts from early phases of the software life cycle.

From the presentation of the different approaches it is clear that domain knowledge, particularly from the efforts to reuse artifacts from analysis, is considered important. Domain knowledge contributes to much of the semantics of the models. This is very much in line with the discussion in chapter 2. Also, most of the efforts have chosen to focus on only one, or perhaps a couple, of the phases of which AR or CBR consist. This is, most likely, due to the inherent complexity of such a task. Much work has been put into the mapping phase of AR, while within CBR more effort has been given to the retrieval phase. To be able to utilize the vast knowledge that is kept within a software model repository, it is important to understand how retrieval can be performed in an efficient way, so the contributions within CBR is therefore important.

In the ROSA project, focus is on the retrieval and mapping phases of analogical reasoning. In this context, CBR is important due to its focus on retrieval, something that has not been as emphasized within AR research. Also, I have discussed genetic algorithms because the AR mapping phase within my project will utilize genetic algorithms. In section 3.6 the work by Tessem on how to use a GA during the mapping of OOram models was described. This work is applicable to my work in a somewhat modified and extended form. In section 3.6, I discussed how these techniques fit into the project.

Chapter 4

Use of WordNet in ROSA

Lexical databases are useful in conjunction with many types of information systems, e.g., retrieval software, and automated natural language translation. In ROSA there is a need to describe components both in terms of their structure and their semantics. I use available results from lexical and semantic research as a basis for describing semantic information of model components. The goal is to identify and organize semantic information in such a way that it can successfully measure the semantic similarity between any two components in the repository.

In this chapter, I look at problems related to how components can be described, and what alternatives exist for solving the problems of identifying and organizing such information. In particular, I look at WordNet (Fellbaum, 1998c) [1]. The goal is to identify properties of WordNet that are useful for finding similarities among OOram components. In section 4.1, I give a brief overview of WordNet, a system that seems like a suitable lexical database to use for this purpose. A full overview of the WordNet lexical database can be found in appendix B. Section 4.2 presents how it can be used within the framework of ROSA, and in section 4.3 I discuss how some additional problems related to naming can be solved.

4.1 The WordNet Lexical Database

WordNet is a lexical database that supports conceptual, rather than merely alphabetical, search (Fellbaum, 1998c). It combines features from both a dictionary and a thesaurus. WordNet con-

[1]A previous version of this chapter can be found in Bjørnestad (1997).

tains the full range of common English vocabulary, and it has been handcrafted and expanded as new sources of lexicons or other lexical information have become available or have been selected. It is organized around meanings in synonym sets, or *synsets*, consisting of all the words that express a common concept, and with relationships between these synsets.

Each *word form*, i.e., the written word, can have several *word meanings*. WordNet, version 1.6, contains approximately 174000 word forms organized in about 91.600 synonym sets. Word forms belong to one of the word categories nouns, verbs, adjectives, or adverbs. Of the word categories, the noun group is by far the largest. The semantic relationships between synsets (examples below are taken from Miller et al. (1993)) are exemplified in appendix B. They include

- hyponymy—specialization—an *elm* is a type of *tree*
- meronymy—part-whole relation, from Greek *meros* (part)—a *treetop* is part of a *tree*
- entailment—implication, i.e., one statement implies others—*John is a bachelor* implies that *John is a man*. Entailment is only used between verbs (see appendix B.3).

Lexical relations can also exist between words rather than between synsets, e.g., antonymy, which expresses degree of oppositeness between words. This relation exists between nouns, e.g., *girl/boy*, between adjectives, e.g., *good/bad*, and between verbs, e.g., *appear/disappear*. These are relations between specific *word meanings*. Although opposite meanings can be identified among words other than direct antonyms, people are not likely to use them together in pairs. While *fall/rise* and *ascend/descend* are direct antonyms, *rise/descend* and *ascend/fall* are conceptually opposed, but they are not direct antonyms.

WordNet does not contain organizational units larger than words, but the relational semantics does reflect some of the structure of frame semantics, e.g., WordNet does relate words like *buy* and *sell* through the antonym relation, as in a frame for a *commercial transaction*; see Fillmore and Atkins (1992). According to Fellbaum (1998a), WordNet “relates words and concepts from a common semantic domain”. The word categories are discussed separately in appendix B.

The basic relation used in WordNet is *synonymy*. Words are organized in synonym sets, or synsets for short. This implies that the words in a synset are interchangeable in some, although not in all, contexts. Some short phrases, or collocations, that would normally not be included in a dictionary, are included in WordNet. Fellbaum says that the natural distinction between *word* knowledge, which would be stored in a dictionary, and *world* knowledge, which would be stored in an encyclopedia, has not been enforced completely. The boundary is fuzzy.

4.2 Semantics in ROSA

In this section, I look at ways semantic information can be used in ROSA to reduce ambiguity and give better analogies. It requires that semantic knowledge is available in a *term space*, or *semantic net*, and that it is organized in such a way that information about semantic similarity between two terms can be identified. Although the term space does not need to contain all possible words or phrases, it should contain all terms that are relevant for the domains of interest, and it should have defined the necessary relationships between these terms. Two example models are described in section 4.2.3. In chapter 5 I discuss similarity measures and algorithms that may help identify good matches during retrieval. Here I discuss semantic relations that may be useful for ROSA; also see Bjørnestad (1997).

4.2.1 Semantic Requirements of the ROSA Analogy Machine

Research in AR emphasizes deeper, structural similarities as important for finding analogies (Gentner, 1983). Surface similarities like the names used, however, may play an important role for improving the analogies. An analogy machine should quickly be able to identify the models that most closely resemble the structure of the *target* model. Tests performed on synthetic graphs, and not on OOram models, using a neural net approach (Ellingsen, 1997b) and genetic algorithms (Tessem, 1998b), indicate that when using additional linguistic knowledge, the search for good analogies is improved. These experiments used names chosen from a restricted set, and the graphs were not conversions of real role models. The names used in these tests were just text with no relations through a semantic net. It was therefore only possible to say whether names were equal or not. One could assume that the addition of a semantic net would enhance the importance of using semantics as an aid to identify analogies. Therefore, the results after searching for structural similarities should be combined with a search for semantic similarity.

Software component libraries are often categorized or classified according to the intended use of each component, and this can often be seen through the names given to the components, e.g., containers are components that are used to store other components. There are different ways to think of most components, however, and several different perspectives should be permitted. The components in my system, e.g., OOram role models, roles, and messages, can be categorized in a set of different ways. These different dimensions may be used to strengthen or weaken an analogy. A set of dimensions might be useful for describing OOram components. For example,

- all components, e.g., roles and methods, have a *name*
- all models have one or several *stimulus roles* that may initiate an activity in the model
- models are described by their *structure*, and the structure descriptions are generated using the stimulus role as the root node in a graph
- For each role in the model, the names of the roles it *knows_about*, i.e., the ones it can communicate with (possible values are zero, one, or many)
- For each port, a role may have a set of *messages* that it can send to the connected role

These dimensions indicate how structural and semantic information identified in models can be combined to find potential analogies. In the following, I discuss the semantics in OOram role models.

4.2.2 The Semantics in OOram models

The study of relations in the context of lexicons may be focused on language understanding, translation, or generation. A system that handles natural languages within any of these areas must handle intended ambiguity and use of metaphors. During analysis of software systems, we try to remove ambiguity and make model names, descriptions, and relations as simple and clear as possible. This will particularly reduce complexity concerning morphological relations. Plural forms for nouns will typically be used only for roles representing collections, or plurality is shown only indirectly through the use of multiplicity of the ports. The last alternative may be preferable because of its simplicity.

Structural ambiguity is, according to Sowa (1991), a problem in natural language. This problem is reduced when role models are created. *Lexical ambiguity*, which arises when a word form can have several meanings, is reduced if the name of a component, e.g., an OOram message, is linked to the word meaning rather than to the word form itself. Each OOram role model represents one part of the problem domain, and some dimensions are explicitly present in it, e.g.,

- *stimulus role*, *structure*, and *contained roles* for the role model
- *name*, *knows_about*, *message sends*, and *attributes* for roles
- *name*, *arguments*, and *return values* for messages

Each role-to-role relation can be expressed using a simple sentence, e.g., *A Borrower returns (a Book) to the Library*. OOram models are examples of what Chaffin and Herrmann (1988) call complex relations made by adding specifications to simple relations or concatenating simple relations. They mention 5 types of case relations, of which three can be found in a simple role-to-role relationship. When we make a role model based on a sentence like the one above, these relationships are made explicit. The *stimulus role* becomes the actor in the given relationship. Taking the model that can be made from the above mentioned sentence, we find:

- agent—action (Borrower—returns (Book))
- action—recipient (returns (Book) to—Library)
- action—instrument (returns—a Book)

Even in a model containing only two roles, each role may be able to send more than one message to the other role. In OOram role models, the two roles represents the *agent* and *recipient*, respectively, and the method can be considered the *action*. The communication between two roles can therefore not be represented by one of the relations mentioned above, which is rather the relation between a role and a message, a message and a recipient role, or a message and an argument. When one role communicates with another, this is in OOram called a *knows_about* relationship. This relationship has some similarities to what Booch (1994) calls an *association*. The difference is that while an association has no direction, the *knows_about* relation is directed.

According to Booch (1994), a relationship first identified as an association may later be expressed in a more precise way, e.g., as aggregation (i.e., meronymy), inheritance (i.e., hyponymy), using (for example in a method or as a parameter in a message), instantiation (of a type or class), and delegation (an alternative to inheritance). The *knows_about* relationship, however, is not applicable in all these situations.

- *Inheritance* is not seen in role models, but one could think of it as modelled as semantic relationships in the term space. In OOram these relationships are delayed to the design and implementation phases, where we talk about specialization of types or classes.
- *Instantiation*—more related to the design phase, and is not discussed here.
- The *using* relationship in the meaning "used as a parameter to a message" is modelled differently in OOram. If "a role is used in a method", this will be modelled by synthesizing a sub-model inside another model or by an attribute of a role.
- The *knows_about* relationships might be substituted with aggregation or delegation.

An alternative way to specialize the *knows_about* relationship is by analyzing the messages a role can send. A message is typically represented by a verb phrase. Generally, verbs are more polysemous than nouns, and thus harder to study (Fellbaum, 1993). The meaning of a verb is to a great extent determined by the noun it is used together with. Fellbaum (1998b) indicates that this problem may be solved by adding a pointer from a verb synset to a noun or noun synset. This will determine in what context the different verb meanings can be used. In OOram models, however, messages are connected to the roles that send and receive them, so this is already part of the model.

4.2.3 Example Role Models

In the following, two role models are shown. One is from a library example and the other from a wholesaler example. They enable us to describe a set of useful relations. I am considering semantic relations between verbs as well as relations that can be seen between roles and messages in the models. WordNet is used to find additional relations. The two models are considered analogous—see discussions in Bjørnestad, Tessem, Tornes, and Steine-Eriksen (1994). Still, the terms used to describe them are partly different. The two models are shown in figures 4.1 and 4.2.

`Borrower` and `customer` are, respectively, *stimulus roles* in the two examples. Both models contains a role called `register`. The multiplicity on the path from these roles to the

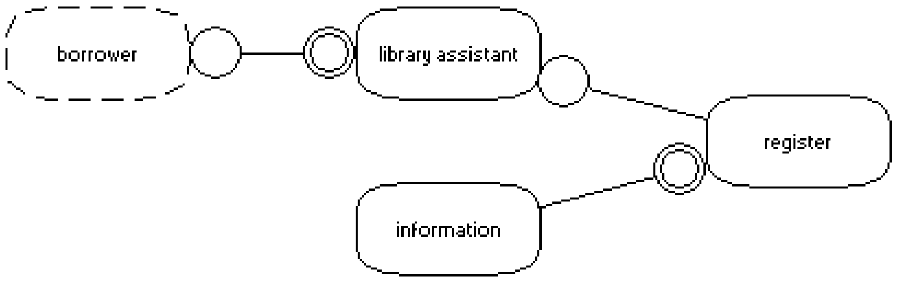


Figure 4.1: An OOram model for a library example where a borrower lends books (This model is named Library_sub0 in Table F.2)

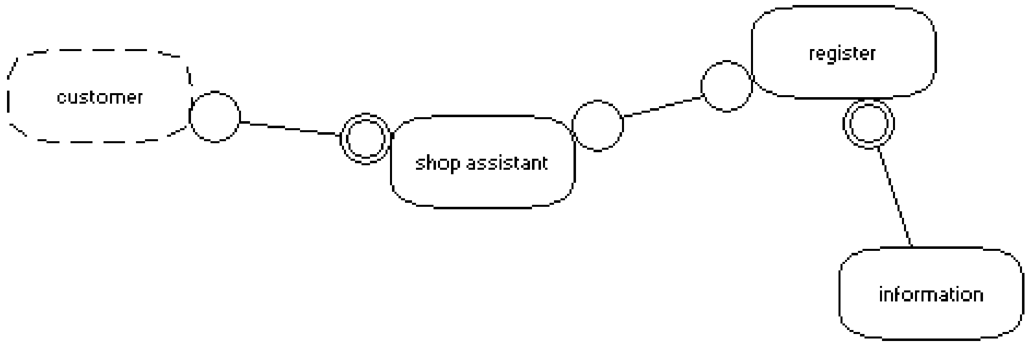


Figure 4.2: An OOram model for a wholesaler example where a customer buys items (This model is named Wholesaler_Sub1b in Table F.2)

contained role indicates that they are containers. A multiplicity of *many*, however, does not necessarily imply containment, as the relation between `shop assistant` and `customer` exemplifies. Some relations between roles in the models are shown below, written on the form *fromRole, predicate, object(), toRole*.

- Borrower, requests, book (“Garp’s Book”), Library assistant
- Library assistant, searches for, Book (“Garp’s Book”), Register
- Register, asks for, name (“Garp’s Book”), Information
- Customer, orders, partID (“carburator”), Shop assistant
- Shop assistant, searches for, partID (“carburator”), Register
- Register, asks for, name (“carburettor”), Information

The predicates are registered in the model as messages. The parentheses represent message arguments to identify a particular instance. Below is a list of relations identified as important, based on the above examples. The two examples are similar in terms of the predicates discussed above, but the analogy breaks when we look at a borrower returning a book. There is no similar situation when it comes to items bought in a store, except for a situation when the item is returned due to a production flaw.

- **Synonymy**—{*person, individual, somebody, someone, . . .* } a synset given in WordNet.
- **Hyponymy** (specialization)—*borrower, customer, library assistant,* and *shop assistant* are all hyponyms of the synset listed above, even though the two types of assistants do not exist in the current version of WordNet. *Assistant* can be found at level 5 in the same hierarchy as *borrower* and *customer* are found. How close they are to each other in this hierarchy can indicate how close they are semantically. Even this simple categorization may help to show similarity.
- **Meronymy** (part-of)—roles may have attributes that also have names from the term space. They may be used in later phases of matching to select the best match. {*course, course of study, course of instruction, class*}—has the meronyms {*lesson*} and {*lecture, lecturing*}.
- **Antonymy** (opposites)—*borrower/lender*, customer does not have a direct antonym. Since only a few terms have direct antonyms, there is a weaker relation called opposites, where other terms in the synsets of the terms being compared have an antonymy relationship.
- **Instantiation**—“Garp’s book” is an instance of a *book*.

Typical antonyms among nouns, e.g., {*man/woman*}, when used in two models that are compared, will not exclude each other as being similar. In addition to being antonyms, they also have a close similarity in WordNet. Therefore, I do not find it worthwhile to include this relationship in the ROSA term space. When searching WordNet for the term *return*, 16 verb senses are found, of which only one is relevant for the example *Returning a book*, sense 2 {*render,return*—(give back)}[2]. WordNet also contains examples of how a verb can be used, e.g., *Somebody __s something* where *return* should be inserted into the open slot as in *Borrower returns book*.

The structure of the two models is similar, and the following mappings can be identified[3]:

[2]Version 1.6 of WordNet is used for the examples in this thesis.

[3]Notice that when selecting terms for a role model, Reenskaug, Wold, and Lehne (1996) recommends that

- borrower—customer
- library assistant—shop assistant
- register—register
- information—information

For example, books are borrowed at the library, while parts are bought at the store. Even though the two models are analogous, not all relations and properties match. When considering other parts of the model, we see that while books are returned after they have been read, parts are not. This difference is not seen in the structure of the models; it is implied by the messages that can be sent. This can be viewed as additional structure that may be used during later stages in the analogical reasoning process. In many situations such differences can not be resolved automatically, and the systems developer must evaluate the suggested mapping of role models. This is illustrated in figure 4.3 below.

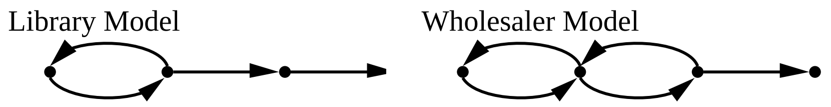


Figure 4.3: The structure of the two role models

I will return to this example during the discussion of the similarity model in chapter 5.

4.2.4 Discussion

WordNet seems to have enough power to describe the semantics of role models, both when it comes to the types of information it contains, the types of relationships that exist between them, and the size of the database. Nouns can be used to name OOram roles. Verbs are used to name OOram messages, possibly in the combination with a modifier, which in this case may be a noun, as in *add name*. The *function facet* described in the classification scheme of Lung and Urban can be compared to the *message feature* of a role in a role model. Role attributes can be named using nouns as well. If messages have arguments consisting of names and types, these may also be named using nouns. If we want to specialize role names, modifiers can be used. These can be either nouns or adjectives.

one use terms that are slightly more general than necessary. I want to add that, if at all possible, it is advantageous to use words that already exist in the repository.

To find similarities between OOram components, e.g., roles and messages, the relationships between words and synonym sets can be used to find how similar any two names are. It is more unclear how attributes and adverbs will be used.

When representing the term space as a semantic net, the relations between the terms represent some kind of underlying semantic structure. The names of the roles or other OOram components, therefore, represent something more than mere surface similarity.

It is unnecessary to store details on morphological forms in the ROSA term space. Plurality is shown indirectly through the use of multiplicity at the ports. Verbs are typically used in present form in models.

In section 4.3, I discuss additional issues related to how to solve specific problems of handling semantics in ROSA.

4.3 Solving Semantic Similarity Problems in ROSA

Problems related to the use of semantic information for identifying analogous role models were described in section 1.3. In this section, I analyze these problems, and for each topic I refer back to the problem in section 1.4 and, whenever relevant, to experiments in chapter 7. If semantic information is to be used to identify similarities between components in a repository of reusable software artifacts, several problems should be addressed. These include

- how semantic information is represented,
- how semantic information is identified,
- how relationships within a repository of semantic information is defined,
- how relationships between potentially reusable components and their semantic description are organized, and
- what semantic information is useful to help identifying possible reusable components.

The goal is to find a representation that is adequate for solving the problem, yet does not require too much work during initialization, information systems modelling, and retrieval. The organization must support the search for similarities, and it must be adjusted to the requirements of the analogy machine. The identification of terms used to name components should not involve too much work on the part of the systems developer. Also, the repository must contain a vocabulary large enough for the user to realistically be able to name the components.

Problems related to naming ambiguity are discussed in section 4.3.1. In section 4.3.2, I discuss the choice of vocabulary for ROSA. Section 4.3.3 discusses alternatives for how to handle situations where several words co-occur, and, in section 4.3.4, I describe how the term space can be filled in an efficient way. Section 4.3.5 presents some related work, and finally, in section 4.4, I conclude about what this means for the ROSA project.

4.3.1 Naming Ambiguity in Software Development

Intuitively we use an artifact's name when we compare it to similar artifacts. When looking for analogies, however, as pointed out in section 3.1, the deeper, structural relationships of the problem are important. In such situations, it is still useful to take advantage of semantic similarities that may exist among components.

Even when not using AR, it is naïve to think that we can use equality between the names people give artifacts as the main criteria for retrieving reusable components. This is due to the fact that people tend to use different words when describing the same thing, depending on their background, or simply because there are several suitable terms to choose from. The problems we face may be related to synonymy, the view the developer takes when addressing the problem at hand, or to differences in abstraction level that is used when describing a component.

One solution to this problem is to require that developers use a restricted language, where they are required to look up terms in a data dictionary to find the "correct" term according to some standard. This may work within a small organization, or when working within a well-understood application domain, but it is hardly adequate for inter-organizational retrieval and less understood domains. Such a restricted language should, if at all possible, be based on domain analysis with a well established ontology for the problem area. Another solution is to relax the requirement of equality, and use some kind of similarity measure. This is the approach discussed in the next section.

During information systems analysis, systems developers normally use some kind of semi-formal approach to model applications. They typically use terms from the application domain, which describe the objects in question, during this initial phase. The selection of terms will, however, depend on the person's knowledge of the domain, and on his or her experience as an information systems developer. For other people it will not always be obvious which of several meanings a developer had in mind. If a domain is less well-known, there is less chance that an established ontology for the domain exists. Introduction of a pattern language for a particular domain could create such a common language for a community of people.

4.3.2 Choice of Vocabulary

In section 4.1, I mentioned that in some situations short phrases are added to WordNet in addition to single words. When people model information systems, compound words are often used to describe components, e.g., roles. In an application such as ROSA, therefore, it may be necessary to use more phrases than what exists in WordNet. This may better reflect knowledge that is useful for describing models within a particular domain. In figures 4.1 and 4.2 we see that the names `library assistant` and `shop assistant` are used, although they are not originally part of WordNet. In the previous prototype, described briefly in section 1.2.3, the addition of new compound terms was made frequently. The user could add new phrases to the database, but it could be difficult to decide where in the term hierarchy new phrases should be linked up.

There are reasons to believe that unrestricted additions of new terms should be discouraged. This will lead to uncontrolled growth of the term space, leading to increased problems when it comes to term selection. This problem should therefore find another type of solution, and this is one of the topics that I hope to shed more light upon during this work.

WordNet has a richer vocabulary within a few technical domains, e.g. medicine and biology. This is because WordNet has been used to perform experiments with regard to people's knowledge within these fields. This is an indication that there is a need for a richer vocabulary within other technical domains as well, and many of the domains that are candidates for information systems analysis are of this type.

Messages are organized in a hierarchy to distinguish between their semantics. Word-

Net organizes verbs in 15 categories according to semantic domains, e.g., verbs of change, communication, competition, consumption, contact, creation, emotion, motion, perception, possession, and social interaction; see appendix B.3. Most of these groups denote events or actions. Another group contains verbs referring to state. These categories should be examined to see how they can be used to structure the relation between messages. Maybe the most important thing when it comes to messages is that they are structured into categories. Fellbaum states that the sense of a verb is often determined by the noun it is used together with.

The new ROSA prototype will use a collection of nouns for describing role names. Verbs can describe messages. In many situations, however, some kind of modifier will probably be needed. An example of a message name is *set*. For one role there may be different types of values that can be set, and we would need different types of messages to set different types of values. If we have a role called `customer record`, we could need the messages

- set name
- set address
- set telephone number

These arguments could be used to set the values of different attributes of the `customer record`. In object-oriented programming, such functionality requires different names for different *set* methods (in this case), to be able to tell what to set when such a message is sent to the same object. One way to solve this problem would involve the use of an argument, but in object-oriented design it is typically not done in such a way, and a modification would therefore be needed before design. If we use name, address, etc., as arguments to the *set* methods, the method names would have to be translated during design. On the other hand, if we had different message names, there would not be a translation problem.

This second alternative may be preferable to doing modification in the term base, or adding complicating constructs in the OOram repository. If implementing the ideas in relation to co-occurrence of words, as described in section 4.3.3 for role names, however, this could just as well be done for messages. Actually, the construct linking an OOram name to a word meaning is done in **OOramNamedComponent**, which is the direct superclass of both **OOramRole** and **OOramMessage** (see figure 6.6 on page 123).

Nouns can also be used to describe the roles' attributes. This possibility has not yet been taken advantage of in ROSA, although in a full-fledged repository this feature may be important.

4.3.3 Use of Word Collocations

Often, when systems developers describe components in a system, they use phrases or collocations. Examples are the role names `area manager`, `section manager`, and `technical expert` (taken from an example in (Reenskaug et al., 1996, p. 151)). These phrases are not part of WordNet, and I discuss alternative ways to approach this problem. Other examples are shown in the example in 4.2.3, e.g., `library assistant` and `shop assistant`. There exist several choices as to how to resolve this problem:

1. Include such terms in WordNet, and decide where in the hierarchy they best fit. This approach was chosen in my first experiments. This approach is not without problems. The term space will grow, and there is no guarantee that the selected hyponymy relations would have been approved by a linguist. One could require that additions of new terms would be controlled by a *librarian/linguist*. In most organizations, however, it is unrealistic to expect that they have a linguist on the team.

2. Simplify the models and select slightly more general terms than would be natural at first. This approach is recommended by Reenskaug et al. (1996) to get more reusable solutions. In some situations this is not a solution. In the example referenced above, where you have several types of managers, they can not all be replaced by the name `manager`. Of course, one of them could be replaced by a synonym, but that may change part of the semantics in the model, and in many situations there are no synonyms. So this is a solution that should be used in some situations, but cannot be applied universally.

3. Permit a component, like a role, to have a name that is composed of several words, where each of them may be connected separately to a component in the term base. Although this is a slightly more complex solution, it permits more flexibility without the problem of a term space with increasing size. Names can be compared without necessarily being stored in the term space. The developer can use meaningful words, yet prevent the term

base to grow uncontrolledly. Allowing a name to be linked to more than one word in the term base reduces the problem of the user having to *insert* new collocations.

Within this approach, there are two alternatives. If a role name consists of two or more words, and the combination is not an existing term in WordNet,

(a) each word is connected individually to a word meaning in the term base. When looking for similarity, the term base is searched for each of them, and the similarity measure must be based on a combination of the individual similarity values.

(b) only the most significant word is actually coded, i.e., the user has selected a word meaning for it. The combined term is then regarded as a specialization of this term, and the similarity value is adjusted for this extra distance in the term space.

Using approach (b), if a role named `area manager`, which does not exist in WordNet, is used, then it is linked to `manager`. As `manager` is not the complete name, the system treats the name as a specialization of `manager`, although it is not stored as such. This approach permits roles to have different names, even if the significant term may belong to the same synset. If two roles are compared, and they both have `manager` as their significant name, but with different modifiers, we know that they are similar, and that each of them is given an extra level in the hierarchy due to the modifier. But they cannot be distinguished between, unless we can somehow compare the less significant words. A simple solution to this problem is to link the insignificant part of the name to its word form. The user does not have to select the meaning. If the two roles are compared and the significant words are equal, the non-significant word are compared. If they are also equal, the two roles have equal names; if not, the names are different, although quite similar. Only when the more significant words have identical word meaning do we have to compare the less significant words.

If I choose the last alternative, we have the following situation: Instead of letting an **OOramNamedComponent** have a name attribute that is a reference to a **WordMeaning**, it should have a list of such references. In many cases, this list will contain only one member. Then this is of course the most significant word. In cases where there are several words in co-occurrence, they are stored separate references to **WordMeaning**s. As for now, I assume that the last of these words will always be the most significant and the one that is used during analogical retrieval and analogical mapping. I then assume that the occurrence of the extra term just adds one level to the synset path. Only when the significant words that are compared belong to the same synset is the less significant word examined. The similarity value for these words must be modified by some kind of weighting. This approach requires, however, more resources to calculate semantic similarity. Particularly during the retrieval phase, where a target model is compared to all base models, such a cost should be evaluated against its benefit.

4.3.4 Filling the Term Space

A problem when developing a repository, such as the ROSA repository, is to fill it with a substantial amount of data so it is possible to test the selected approach. One such problem is the organization of words and definitions of the relationships among them. A software developer, who is not also a lexicographer, cannot fill a database with lexical and semantic information and claim that it is correct. WordNet seems to solve the problem of how terms should be organized, and I will adopt this lexical database.

Another problem is actually to fill the database. WordNet solves this problem, too. The lexical information of WordNet is also distributed as a set of Prolog files, where each file contains one relation, e.g., elements from synonym sets denoting what type of word it was (e.g., noun, verb), synset glosses, hyponyms, antonyms, and so on. These files can be read and interpreted in a specified sequence, and the term space can thus be filled with the information that is already organized by experts. I have currently not incorporated all word categories and all the relations of WordNet into the term space. The limitation is how to utilize it rather than the availability of data. The word categories and relationship types may be extended as it becomes more clear how the different types of information may be used in ROSA. An important point related to what should be included in the database is that addition of too much information may slow down the analogy machine, so additional word categories and relationship types should be added with caution and only when it is clear how it will be used.

4.3.5 Related Work

The project *ROSA* (**R**euse **o**f **S**oftware **A**rtifacts) (Girardi and Ibrahim, 1994), not to be confused with our own project, uses WordNet to obtain morphological information, grammatical categories of terms, and lexical relationships between terms.

Burg and van de Riet (1998) use WordNet in *COLOR-X*, a linguistically based conceptual modelling environment. WordNet is used during conceptual modelling to ensure that the resulting models are correct. WordNet is also used to paraphrase models into natural language sentences. The models that are made are static object models and event models. Burg and van de Riet claim that WordNet supports conceptual modelling. Information from WordNet is used to select meanings of polysemous words. Once the correct meaning has been selected, they extract information from WordNet about *substance*, *part*, and *member* meronymy relationships. They augmented WordNet with more relations, e.g., function words, and thematic roles, and they also had to enrich the morphology.

Gulla, van der Vos, and Thiel (1997) use a linguistic approach to retrieve conceptual models. Using their suggested approach, a user can search a repository using natural language, since he may not know the syntax of the modelling language that is applied. They use a model where the grammar-relevant semantic properties of words and phrases are modelled as conceptual structures in a sign-expansion framework. The lexicon assumed in this model is made up of conceptual frames that are not associated with any word category, but can be expanded in different directions to support different words and different sub-categorization frames.

Faceted classification of models is criticized by Gulla et al. (1997) because this classification does not really consider the semantics of the search terms (facet values). They also criticize approaches where the semantic structure between elements of the model, e.g., between a *loan* and a *book*, is not analyzed, but where you can analyze each word by itself for synonyms, hyponyms, and meronyms. Due to this problem the following problem can occur: The above criticized approach will be able to capture the similarity between a book reservation system and a dissertation reservation system, but not between any of these and a car reservation system. The reason for this is that it is not the relationship between the book and dissertation, or car, that is interesting, but the structural relationship between *loan* and *book* or between *loan* and *car*. I discuss this example further in section 5.6. An example of the approach they criticize is Richardson and Smeaton (1995), who have used WordNet as the lexicon for an information retrieval system.

4.4 Discussion

In this chapter, I have discussed WordNet and in what way it can be useful as the basis for the ROSA term space. The information from WordNet that will be included in the term space is only what is directly useful for the AR machine. First of all this is the noun part that can be used to name OOram roles. Later, verbs will be included to name messages, typically with nouns as modifiers. The design of the OOram components should be flexible so that this later extension is made easy.

The criticism from Gulla et al. (1997) of the work of Richardson and Smeaton (1995), mentioned above, is concerned with their lack of structural information about the model itself. This criticism is not really a problem for my approach, since I do not query the database in a natural language, but rather in a semi-formal language in the form of a role model; see section 4.2.2. The structure of the OOram model is used as a search criterion in addition to the semantic information of the names of roles and messages in the model. This structural information is of a similar type, although formulated in a different way, that they refer to.

Since I use role modelling instead of a natural language to describe role models, part of the enhancement that Burg and van de Riet (1998) suggests, e.g., thematic roles, is not needed. The algorithms used to identify similarities during retrieval and mapping is discussed in chapter 5, while the design of the term space is described in chapter 6.

Chapter 5

The ROSA Similarity Model

Similarity is, in McKechnie (1979), defined as "1) having characteristics in common: strictly comparable, 2) alike in substance or essentials, 3) not differing in shape but only in size or position". In AR, emphasis is put on finding deeper, structural similarities between a target case and one or several base cases. Owen (1990) argues, however, that structure alone is not enough to distinguish among cases that are similar and those that are not. I propose a similarity model based on a combined measure of structural and semantic belief values both during the retrieval and mapping phases. This measure is fine-tuned based on heuristics discussed in this chapter.

In section 5.1, I outline a few earlier approaches to defining similarity models for identifying reusable software components. In section 5.2, the overall similarity model proposed in this thesis project is presented, and I describe how it is applied during retrieval and mapping. Similarity considerations are handled in two ways:

1. during retrieval—to a great extent based on semantics
2. during mapping—based more on structural information.

Algorithms used for defining structural similarity are based on structure descriptions as described by Tessem (1998a), and use of genetic algorithms (Tessem, 1998b) as discussed in section 3.6. These algorithms are here adapted for use with the ROSA repository, and particularly with the ROSA term space.

My discussion of the similarity model has its main focus on semantic similarity, and two alternative semantic similarity models are presented in section 5.3. The motivation for suggesting several versions of how to measure similarity is to study how different variables influence the result. The models are tested individually, and a choice is made as to what model is best suited for the ROSA project. In section 5.3.4 the similarity models are compared using examples from a WordNet hierarchy. Section 5.4 gives a practical example of how similarity is calculated both during retrieval and mapping, and how the two semantic similarity models behave when comparing the role models shown in figure 1.4.

Section 5.5 discusses the possible advantage of distinguishing between ONE and MANY ports in role models when mapping analogies. The use of such information may help to improve mappings through a more complex approach to similarity. Finally, in section 5.6, I discuss the selected approach in relation to the approaches presented in section 5.1.

5.1 Previous Software Component Similarity Models

Several similarity models have been proposed to identify reusable artifacts within software development. Each is suggested within the context of a given research project with a specific type of artifact, level of abstraction, and system development methodology used, and they may therefore be difficult to compare. I therefore describe the context within which they have been suggested.

5.1.1 The AIRS System

Ostertag, Hendler, Díaz, and Braun (1992) discuss a technique to compute similarity among components in a reuse library system (see section 3.4.1). The technique represents a formalization of the concepts on which AIRS (AI based Reuse System) is based. The functionality of a prototype implementation of AIRS is illustrated by the application of two different software libraries: a set of Ada packages for data structure manipulation, and a set of C components for use in Command, Control, and Information systems. This can be categorized as reuse of code. These authors also seem to be focused on the mapping phase only.

A component is described by a set of (*feature, term*) pairs. A feature represents a classification criterion and is defined by a set of related terms. Candidate reuse components are selected based on the degree of similarity between their descriptions and a given target description. Similarity is quantified by a nonnegative distance proportional to the effort required to obtain the target given a candidate. Distances are computed by comparator functions based on subsumption, closeness, and relations.

Ostertag et al. set the distance between a term and its synonyms to zero. Therefore, a component for which the synonym is used will come up as a good candidate. There seems, however, to be no attempts to handle polysemy, i.e., the situation where one term can have several meanings, as WordNet does.

5.1.2 Analogical Similarity of Objects—Conceptual Modelling

Spanoudakis and Constantopoulos (1992) define similarity between software artifacts and discuss its potential exploitation in software reuse by analogy. They focus on the elaboration phase of AR. They first identify what properties are important for finding similarity. Thereafter they develop a systematic basis for comparison within a general conceptual modelling framework. Finally, a general form of distance metrics for computation of similarity measures is defined.

The conceptual distance between two objects is computed as an aggregate of partial distance metrics defined over the representation dimensions of the Telos language, namely identification, classification, generalization, and attribution. They are described as follows:

- The *identification distance* distinguishes between identical and non-identical objects even if the latter are equal, i.e., they belong to exactly the same classes; they are generalized to exactly the same superclasses; and they have attributes with equal values.

- The *classification distance* reflects differences expressed by the classification of two objects in different classes. The non-common classes of the objects have different contributions to the classification distance depending on the relative importance of the categorization they represent. The relative importance of each class is a decreasing function of the depth of the class in the taxonomies (generalization/specialization graphs) in which it participates.

- The *generalization distance* reflects differences as captured by the assignment of non-common superclasses. Like non-common classes in the case of the classification distance, non-common superclasses have different contributions to the generalization distance which are similarly determined.

- The *attribution distance* measures differences that occur at the level of attribute assignment. Computing this distance involves establishing a (partial) one-to-one relationship between the attributes of the two objects, which yields corresponding and unique attributes for each object. The notion of corresponding attributes is an important and useful extension to the notion of common attributes: they are attributes that can be considered comparable regardless of whether they bear the same or different names and modelling. A necessary condition for two attributes to be comparable is that they belong to common attribute classes (*the principle of semantic homogeneity*).

5.1.3 Systematic Software Reuse Through Analogical Reasoning

Whitehurst (1995) uses analogical reasoning to support reuse of software frameworks, i.e., the classes that constitute such frameworks. This may be called reuse of design. The reuse strategy was described in section 3.2.2. In this section the similarity measures will be described.

When a programmer creates a new association, he must assign a heuristic estimate of the extent to which this new association is related to a particular association group. Groups are similarly categorized into conceptual groups, and so on, forming a network of conceptual distances. So, if $a_0, a_1, \ldots, a_n$ denote the associations corresponding to all vertices in the shortest path from a_0 to a_n, and $\delta(\overline{\alpha\beta})$ denote the cost of the path segment from α to β, the *conceptual distance,* Δ, from a_0 to a_n is given by:

$$\Delta(a_0, a_n) = \sum_{i=0}^{n-1} \delta(\overline{a_i a_{i+1}})$$

Systemic difference is the result of a pairwise application of a modified structure mapping algorithm, pre-computed for associations in the software repository without regard to the resulting mappings. This information is stored in a table with one row and one column for each association added. Relational indexing is information about which frameworks reference a particular association. Relational indexing serves as the basis for deriving a set of similar frameworks from a partial framework specification through the computation of a relational coverage metric. Relational coverage is a measure of the extent to which a particular framework matches the association components of a partial framework specification.

A partial system specification is given in the form of object classes and associations. This specification is tried mapped to a set of source frameworks. Augmenting the matching procedure with feature recognition or other heuristics has the potential of changing the reasoning process from analogy to some other form of similarity, the result of which is undefined. In Whitehurst's work the entity relationships are made explicit, thereby reducing the need to augment the representation with additional feature-based information. The relationships are also related through abstractions, and it is therefore possible to relax the definition of relational equality to that subsumption.

5.1.4 REBOOT

In REBOOT (Reuse by Object-Oriented Techniques) the overall aim is to "provide methods and tools to support creation and use of domain oriented components" (Gronquist, Villermain, and Bongard, 1992). A similarity model for software components is proposed in Karlsson, Sindre, Sørumsgård, and Tryggeseth (1992). Two weights are used to compute the distance between objects. They are

- *term weights*—describe relative importance of terms that classify a component
- *edge weights*—describe the distances between various words

Term Weights

A term is the value of a facet used to classify a component. An estimate is made as to how much of the component a term covers. The weights of terms used to classify a component need not add up to 1.0. When a component is stored, the value of each facet is estimated. For a `stack`, methods like `push` and `pop` are given a high weight, while other, less important methods are given a lower weight.

Term weights are used during search. Terms are search criteria and do not specify actual components The weight given implies the importance for this particular term for user satisfaction. A high value given to a particular term indicates that this term is vital. If no weights are given, the specified terms are given an equal weight of 1.0. A weight can also be given for a facet, indicating its relative importance.

Edge Weights

Edge weights are used to permit relaxed search. There is one term space for each facet. Term spaces have *generalization/specialization* and *synonym* relationships. There is a weight, $w_{t,u}$, assigned to each direction of an edge. A weight is defined as the effort of changing a component classified by the term t to one that could be classified by the term u (E_C), divided by the effort of satisfying the term u from scratch (E_S):

$$w_{t,u} = E_C / E_S$$

Weights are not symmetrical. The effort of changing t to u need not be the same as changing u to t. When terms are not directly connected, $w_{t,u}$ is used to denote the total weight of the path with smallest total weight. A weight, $w_{t,u} > 1.0$, implies that it is uninteresting to modify the part of the object covered by the term. Nothing will be gained from it compared to developing this functionality from scratch.

Each object is classified with a set of weighted terms for each facet. A query has the same format as the classification of objects. The relative distance between a query and every component in the database is found by

$$\min \sum_{facets} w_f \sum_{terms} w_r w_{c,r} w_c$$

where w_f denotes the weight of a facet, w_r the weight of a requirement, and w_c the weight of a component. During search, each w_r is matched against the nearest $w_{c,r}$, i.e., the smallest $w_{c,r}$ independent of w_c and w_r. Weights are estimated based on the *abstraction* and *dependency* facets.

In Sindre and Sørumgård (1993), they recommend building the REBOOT terminology in a *top-down* manner, starting with domain analysis. After a dictionary is built, the terms must be structured into a term space that reflects the relations among them. After this structure is made, the classification process begins. They recommend taking the terminology control task seriously, as the first attempts may not get it right. Their example is concerned with reuse of code, but they do also talk about reuse of domain analysis. The similarity measures are planned to be used during retrieval, but this research is not related to AR.

5.2 Structural and Semantic Similarity in ROSA

This section defines the framework for the semantic similarity model presented in section 5.3. This model is developed for use in an AR context. I first discuss the use of structure descriptions for role models in section 5.2.1. Structural similarities are based on these descriptions. Structural and semantic retrieval are both discussed in section 5.2.2. Finally, I discuss the mapping in section 5.2.3.

5.2.1 Structure Descriptions

A role model is considered a directed graph, where a stimulus role is the root node. Tessem (1998a) describes an algorithm for identifying sub-structures in such graphs. The sub-structures are found based on paths between pairs of nodes along which messages can flow. The number of possible sub-structures that can be extracted from a role model depends on its size and complexity. Tessem classifies them as tree, star, ring, and sequence, listed in decreasing importance. When a new role model is created and added to the repository, its sub-structures are identified and stored.

The idea is to identify, if possible, the three most important sub-structures in each role model. It is assumed that these sub-structures contain the roles of a model that are considered most important for the model's overall behaviour, as they are believed to have the greatest impact on its reuse potential. In addition to a sub-structure's type, its value, or weight, also depends on its size, e.g., trunk length for a tree, number of branches for trees and stars, number of nodes in a ring, and the length of a chain (see figure 5.1). The open circles in the figure denotes stimulus roles and the black dots represents other roles in the models.

In the implementation, information about a role model's sub-structures is saved in an object of type **StructureIndexSet**. The collection of index objects is used during retrieval to identify *base* models that are structurally similar to the *target* model. The more complex the

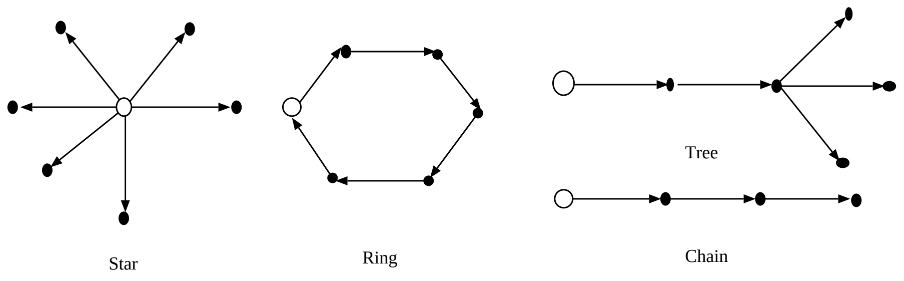


Figure 5.1: Structure abstractions (Tessem and Bjørnestad, 1997)

communication between roles in the model is, the higher importance this part of the model should be given.

5.2.2 Similarity During the Retrieval Phase

The search for potential analogies for a particular *target* model is initiated by doing a search based on the structural index. The result when comparing the index of each base model to the target model is in the form of a *belief value*, and is stored in a record that is inserted into a collection C. The belief value is computed on the basis of the similarity between the structure indices of a base and the target. Each element in C contains 1) reference to a base model, 2) the belief value based on structural similarity, and 3) the belief valued based on semantic similarity.

During the process the elements of C are ranked according to a *probability* that the base model is analogous. The probability is calculated through the evidence found in the two beliefs. Information about beliefs is added during the process, and it is, in the following, denoted differently to underscore the current stage in the AR process. The collection is denoted C_{vac} (vacuous) before any belief functions have been added, C_{str} after belief functions from structural retrieval have been found, and C_{comb} after the structural and semantic belief functions are identified.

In this section I first

- give a brief description of the belief values as understood in the context of the Dempster-Shafer theory of evidence (Shafer, 1976) and how they are used,
- discuss the structural similarity,
- then present semantic similarity,

- and finally, the combination of the results from these retrieval phases is given.

Belief Theory

Analogical similarity is in itself a rather vague concept to which it is difficult to assign a numerical value. In the ROSA project, I use Dempster-Shafer theory of evidence (Shafer, 1976), or belief theory, to convert a value into a *belief function*. Belief theory is used to give a measure of how strong an analogy is. When comparing two role models, there are only two possible outcomes of a test for analogy: there is or there is not a useful analogy between them. Thus, the frame of discernment, Θ, as used in belief theory is given by

$$\Theta = \{a, \neg a\}$$

where a indicates that there is an analogy, and $\neg a$ indicates that it is not an analogy.
The mass assignment function in belief theory is a mapping m

$$m : 2^{\Theta} \rightarrow [0, 1]$$

where 2^{Θ} is the set of all possible subsets of Θ. $m(\emptyset)$ = 0 and $\sum_{A \subset \Theta} m(A) = 1$.
The belief function Bel is then defined by

$$\text{Bel}(A) = \sum_{B \subseteq A} m(B) \tag{5.1}$$

where A can be $\{\emptyset\}, \{a\}, \{\neg a\}, \{a, \neg a\}$. An example of a belief function is given below. The belief that something is either an analogy or not, is always 1.

$$\begin{aligned} \text{Bel}(\{a\}) &= 0.4 \\ \text{Bel}(\{\neg a\}) &= 0.2 \\ \text{Bel}(\{a, \neg a\}) &= 1 \end{aligned} \tag{5.2}$$

In the following I use the term b to denote a *base model*. So, through the use of belief theory I try to establish a belief about whether the base model b is analogous to a target model t or not. The belief functions are given different names depending on whether they are beliefs based on structural, semantic, or combined evidence, as in $\text{Bel}^b_{str}(\{a\})$, $\text{Bel}^b_{sem}(\{a\})$, or $\text{Bel}^b_{comb}(\{a\})$.

The combined belief function that a target model t is analogous to a base model b is found by combining the structure and semantic belief functions as in

$$\text{Bel}^b_{comb}(\{a\}) = \text{Bel}^b_{str}(\{a\}) \oplus \text{Bel}^b_{sem}(\{a\}) \tag{5.3}$$

using the standard belief combination operator $\oplus$ as defined in Shafer (1976).

To conclude the discussion on belief functions used during retrieval, I give an explanation of how the *probability value* based on a belief function is computed, and that is used for ranking base cases. It is this value that is named *retrieval similarity* in other parts of this thesis.

Calculating the probability value can be done after each stage in the retrieval phase, but most important is the combined probability calculated just before the beginning of the mapping phase. The collection C is ranked based on the probability values, and C may be truncated either by selecting a maximum number of base models to be included during the mapping phase or by setting a lower threshold for the probability value (or retrieval similarity). The probability is given by

$$p^b(a) = m(\{a\}) + k \cdot m(\{a, \neg a\}) \tag{5.4}$$

where the value of k decides whether the amount of un-assigned belief mass assigned to $p^b(a)$ is optimistic, pessimistic or neutral; $k = 1$ is optimistic, while $k = 1/2$ is neutral. In the ROSA project there is used a neutral approach. In the following, I will use the shorthand notation p^b to denote $p^b(a)$.

Structural Similarity

When searching for structurally similar models, the structure descriptions described in section 5.2.1 are used. These descriptions are used as indices when searching for analogies, so such an description is also called a structure index. An example of a structure index is: T 3 4, meaning a tree with trunk length 3 and with 4 branches.

For each base, the *target* model's *three* indices are compared to each of the *base* models' three indices. Belief functions indicating similarity between each pair of indices are computed, and the best matchings are selected and combined to give an overall belief for the similarity of the structure descriptions.

The more roles in a role model that are included in selected sub-structure indices, the higher is the similarity between two indices. A model that contains many roles, may have many different sub-structures, and each sub-structure may include only a limited number of these roles. Since only the three most important sub-structures are stored, some structural information may be lost during retrieval, particularly for large models. This implies that the larger, and more complicated, a role model is, the more roles may be excluded from the structure descriptions, and the less likely it is that the best mapping may be found. This is one of the reasons this method is used only for sub-models in the repository. Derived models that are made by combining two or more sub-models, will contain more roles and will have greater complexity.

The results of the structural similarity computations are stored in the elements of C_{str} as belief functions. In the implementation, each element is represented by a record containing a pointer to an index object, $\textsc{Index}_b$, of the base model itself, the beliefs we have in this model being structurally and semantically similar to a given target model, and a probability value calculated based on one of these belief values, or a combined belief value.

The structural similarity belief is denoted Bel^b_{str}, and the probability based on structural similarity is denoted p^b_{str}. This probability value is used as a criterion to sort the list. C_{str} can be sorted based on the probability values and truncated based on either a lower limit of the probability value or the number of elements. A perhaps modified C_{str} is then used as input to the semantic similarity search.

Semantic Similarity

For convenience, I here give a short overview of how semantic similarity is computed. A full description is found in section 5.3.

The semantic similarity value between two roles is based on the distance between the two word meanings representing these roles, and it is measured as discussed in section 5.3. The further apart they are in the term space, the lower the similarity value. For each role in the target model, I select, from the base model, the role with the highest similarity value.

When the highest similarity value for each role in the *target* model is found, I calculate the average similarity value for all the roles in the target model. This value is converted into a belief function, Bel^b_{sem}, and stored in the same structure as Bel^b_{str}. Based on this, a probability value for semantic similarity, p^b_{sem}, is calculated.

Thus I have chosen an *optimistic approach* to computation of similarity between models. For each role in the *target model* the best possible match to any role in the *base model* is found. The assumption is that the best matching role is also the role to be matched in the mapping phase, which of course does not necessarily have to be true. In an analogical mapping, however, two roles are not mapped to the same role in the target model, which may happen in this optimistic model of semantic similarity. The reason for this is that during mapping, only two roles that are in a similar structural position are mapped. What is found during semantic retrieval is thus an *upper bound* for the semantic similarity.

Each role has a name found in the term base. It is not linked to the word itself, but rather to a selected *meaning* of this word. The word meaning belongs to a synonym set with a given synonym set ID. Role names are compared semantically by comparing their synonym set IDs. This is a relatively inexpensive operation. This implies that I consider synonyms to be equal, or to have zero distance, so they can replace each other, which is not entirely true in most cases.

Combined Similarity During Retrieval

As mentioned before, *belief functions* for both structural and semantic similarity between one target model and each of a set of base models are calculated. These belief functions are denoted Bel^b_{sem} and Bel^b_{str} respectively. Based on these, a combined belief value, Bel^b_{comb}, is calculated by equation 5.3. Further, I compute the probabilities p^b_{comb} from Bel^b_{comb}. The results are stored in C_{comb}. C_{comb} is then sorted in decreasing order according to the value of p^b_{comb}. This permits the user to pick the best candidates for the analogical mapping phase based on a lower threshold value for p^b_{comb} or a specific maximum number of base models.

p^b_{comb} is the same number referred to as retrieval similarity in other parts of this thesis.

The two belief functions, Bel^b_{str} and Bel^b_{sem}, in equation 5.3 are given equal weight. It is possible, however, to adjust one or both belief functions in order to change the relative importance given to each of them. At this point, however, there is no indication that either one of these similarities should be given more weight.

5.2.3 Similarity During the Mapping Phase

The best candidates for analogy identified during retrieval are selected for the mapping phase, where their roles are tried to be mapped to the roles of the target model. This mapping results in a similarity measure, also referred to as mapping similarity.

Structural and Semantic Similarity

During retrieval, the textual *structure descriptions* of the models are compared in order to estimate similarity, while during mapping specific roles from target and base models are mapped according to their structural *positions*. The resulting mapping is the basis for computing an analogical similarity between models. The genetic algorithm developed by Tessem (1998b) presented in section 3.6 has been adopted for use with the ROSA repository to identify such mappings.

When the mapping phase is initiated, a matrix is generated representing the connections between nodes in a role model. This matrix contains the value 1 if there is a port—ONE or MANY—between the two nodes and 0 if there is no port; see table 5.1. Such a matrix is created for both the target and base models. The GA has a fitness function that gives score for local isomorphism in the graphs, combined with use of the semantic similarity information between the mapped roles. One example of such an isomorphism from the models in the example is shown in figure 5.2.

	borrower	library assistant	register	information
borrower	0	1	0	0
library assistant	1	0	1	0
register	0	0	0	1
information	0	0	0	0

Table 5.1: Matrix graph representation for the library sub-model

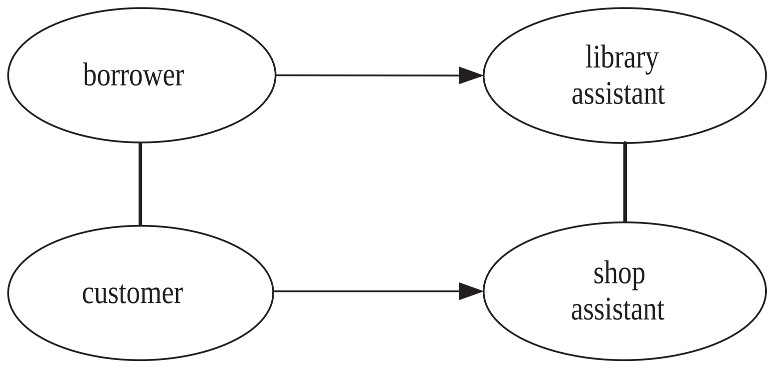


Figure 5.2: Example of an isomorphism from the example graphs

For each pair of role models that are mapped, a matrix is created containing the semantic similarity values between each of the roles in the target and base models. Such a matrix for the two role models shown in figure 1.4 is shown in table 5.2. The fitness function uses the semantic similarity matrix to optimize the mapping.

	borrower	library assistant	register	information
customer	0.8	0.53182	0	0
wholesaler	0.42692	0.40714	0	0
shop assistant	0.53182	0.83333	0	0
register	0	0	1	0
information	0	0	0	1

Table 5.2: A semantic graph example for the role models in figure 1.4 using the semantic similarity model in equation 5.8

The best mapping is taken to be the one that gives the highest total mapping similarity. The analogy value $\text{sim}_A(B_k, T)$ (mapping similarity) is taken to be the average value of the similarities between the mapped roles, i.e.,

$$\text{sim}_A(B_k, T) = \frac{\sum_i \text{sim}(b_i, t_i)}{n} \tag{5.5}$$

where B_k is a base model, T is the target model, b_i is a base role, and t_i is its corresponding role in T given the actual mapping.

The GA algorithm is typically set up to use a large number of individuals and generations (more on this in section 7.2. This implies that for each base model, the algorithm is run many times. This results in a relatively expensive algorithm, and the upper limit of *base* models that are mapped to a *target* model may therefore be set by the systems developer. This is done in one of two ways:

1. setting a lower limit on the probability values, p_{comb}, calculated during retrieval
2. restricting the number of base models selected for mapping

In the first case the value of the lower limit must be based on experiments to find what gives an acceptable analogy. A combination of these two restrictions can also be used. The system developer, however, is the one to decide how this should be restricted.

This algorithm searches for the roles that share the same structural position in the two models being mapped. The best mapping is taken to be the one that gives the highest total value, i.e., it maps as many of the roles in the target model as possible given that this results in a higher similarity value. The genetic algorithm can be fine-tuned through a set of parameters to improve the result. These include the size of the population, the number of generations to be used, and the mutation probability of any individual in the population.

Increase Possible Similarity Between Stimulus Roles

The basic assumption is that when two role models are compared for analogical similarity, the models are treated as directed graphs. I use the stimulus roles as the roots of the two graphs, and compare the two stimulus roles first. To ensure that the algorithm should be more likely to map these two roles, the implementation gives an extra weight to this similarity value as shown in equation 5.6.

$$\text{sim}(t, b) = 0.5 + 0.5 \cdot \text{sim}(w_t, w_b) \tag{5.6}$$

Here sim(w_t, w_b) denotes the used similarity between the two word meanings used for the stimulus roles, and sim(t, b) denotes the resulting similarity between those roles.

Calculated Mapping Similarity

As seen, when a set of structural mappings is identified, the value of the analogy is calculated based on the semantic similarity values for the mapped roles. The algorithm used is similar to the one used for finding semantic similarity during retrieval. The difference is that, while during semantic retrieval the best semantic mappings, disregarding structural position, are chosen, now only roles that are structurally mapped, are compared. This implies that the semantic similarity value may be, and often is, lower than what has been found during the retrieval phase. The algorithm is shown in section 5.3. A mapping between two roles that are structurally mapped, but have no semantic similarity, attributes nothing to the total similarity.

Finally, I calculate the average similarity value of the roles of the target model. This is called the value of the analogy between the two models. Some roles of the target model may not be mapped at all, and the value is adjusted according to this. How this can be tested and solved, will be discussed in section 7.2.

In section 5.5, I discuss how the structural similarity model can be modified due to the multiplicity of roles a particular role *knows about*.

Results from AR do not represent accurate measures, so in the end the user must always interact with the system to accept or reject a resulting analogy. This is done during what is called the *justification* phase of AR.

5.3 The Semantic Similarity Model

This section presents two versions of a model for semantic similarity between two role names, i.e., synsets, of increasing complexity. These models must be tested and, if necessary, modified during experimentation. The goal is to identify a model that, during retrieval,

- will help identify (all) the best analogous role models (high recall)
- will help exclude role models that are not relevant (high precision)
- is simple enough to have an acceptable computational complexity

In other words, all relevant role models should be included among the list of models that are sent to the mapping phase, and this list should preferably exclude irrelevant models. The number of models that are sent to the mapping phase should be as small as possible, but the user should still be confident that all potential candidates are analyzed. The retrieval phase should be executed in as short a time as possible.

It would also be preferable that the base models' values are spread out as much as possible on a scale from 0 to 1 during retrieval to clearly distinguish models that are good candidates for analogy from the ones that are not. The proposed similarity models are therefore tested to see how well the model is able to distinguish between distances between target and base nodes. Such tests are done using spreadsheets. A model that best spreads the word meanings based on their semantic distances is, in this context, considered better than a model that do not.

First, in section 5.3.1, I discuss how the individual roles are compared semantically by comparing synset IDs. Next, I present a simple similarity model in section 5.3.2. This model only takes into consideration the distance between the two synsets that are compared. A slightly more complex similarity model, in which it is taken into consideration that roles with names localized at different abstraction levels in the WordNet hierarchy, should be considered less similar than if they are found on the same abstraction level.

5.3.1 Comparison of Synsets

I have previously mentioned that synsetIDs, and not the word meanings themselves, are compared. In the next section I explain how this comparison is done. Before two role models are compared for semantic similarity, for each model involved in the comparison, a list, IDL_{rm}, containing one list of synsetIDs for each role in the role model, is created. Each of the sublists, IDL_i, is filled with the synsetIDs of the synsets from the synset of the name of role i to the root in the WordNet hierarchy. The root's synsetID is the last element in IDL_i.

5.3.2 Similarity Models

The models presented in this section assumes that each role has been described by a single term, i.e., word form, from the term space, and that, if necessary, a particular meaning of this word form has been selected. These models represent approximations that, in spite of their simplicity, give results that are satisfying. The models do not distinguish between equality of synset and equality of word meaning.

For both models, if the synsets being compared do not belong to the same hierarchy, then $sim(w_t, w_b) = 0$, where w_t and w_b are word meanings. And if the two word meanings are the same, then $sim(w_t, w_b) = 1$.

The first model For the first model, similarity between a role in the target role model and in the base model is equal to the similarity of the word meanings chosen for the role, which is calculated as in equation 5.7:

$$\begin{aligned}
&\text{sim}(w_t, w_b) \\
&\quad = 1 - \frac{(\text{lev}(w_t) - \text{lev}(w_p)) + (\text{lev}(w_b) - \text{lev}(w_p))}{\text{lev}(w_t) + \text{lev}(w_b)} \\
&\quad = \frac{2 * \text{lev}(w_p)}{\text{lev}(w_t) + \text{lev}(w_b)} \qquad (5.7)
\end{aligned}$$

In equation 5.7, w_p is the nearest common ancestor of w_t and w_b and lev(w_x) is the level of a word meaning, w_x, in a WordNet hierarchy. Figure 5.3 illustrates two such trees. The nodes' level is indicated on the left side of the figure.

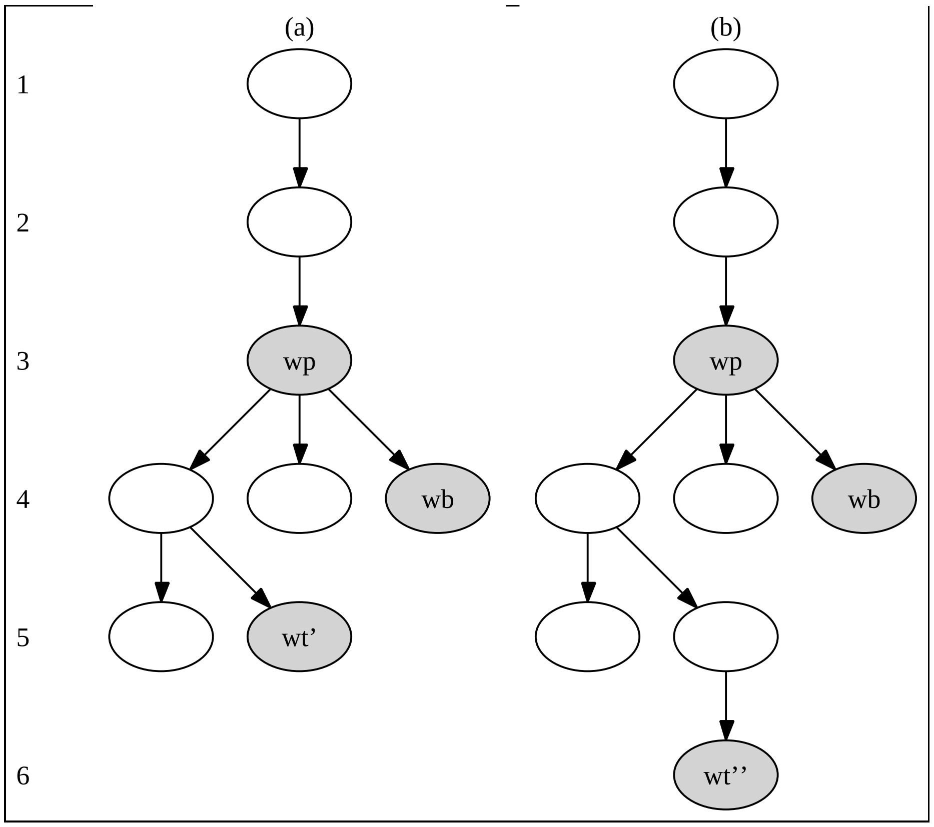


Figure 5.3: Trees with the nodes representing the word meaning of the target (w_t) and base (w_b) and their closest common ancestor (w_p)

In section B.3.1, I stated that in the WordNet hierarchies, there seem to be specific levels in the hierarchies where the number of hyponyms, i.e., specializations, are much higher than at other levels. Words at this level can typically be more easily replaced by a hypernym, i.e., a more general term. At lower levels in the hierarchy—closer to the root—there are often more technical terms that are very seldom used, but which have an important function in organizing the lexicon.

One possible suggestion could be to view the relative semantic distance between synsets dependent on the levels at which they are found in the hierarchy, i.e., their distance from the root synset. Intuitively, one would think that the semantic distance between the synsets

$<$*entity, something*$>$ and $<$*life form, organism, being, living thing*$>$

is greater than the distance between

$<$*clerk*$>$ and $<$*desk clerk, hotel desk clerk, hotel clerk*$>$

The first two synsets are at levels 1 and 2, respectively, while the last two are at levels 6 and 7, respectively.

G.A. Miller, the principal investigator on WordNet, states that he believes "that the distance between two lexicalized concepts is an inverse function of the number of linguistic contexts in which they can be interchanged" (Miller, 2000). He does not, however, have any faith in node counting as a measure of semantic distance. He says that "if you believe, say, that *clerk* and *desk_clerk* share more contexts than do *entity* and *organism*, which seems very plausible to me, then the more peripheral would be more similar than would the most generic". He is reluctant to say anything general about this, and states that there would be great differences among the different word hierarchies.

Such a difference in similarity is, however, somewhat supported by Fellbaum (1998b, p. 72) in her discussion on verbs. She argues that the relative distance between *communicate* and *chat*, and between *do* and *communicate* is not the same. While the first two words are so close semantically that they might be interchanged in some situations, the last two words are not. This may support my suggestion to make the level of the tree significant for calculating the similarity between two synsets.

Notice that this argument is taken care of in the described similarity model. This can bee seen by observing that even though the distances between the word meanings and their common ancestor is constant, the similarity increases if the word meanings have higher levels in the tree.

The second model Whereas the first model may give quite good results, there is another factor that must be taken into consideration. This is the relative difference in height between the two synsets representing the two roles being compared. The greater the difference in height, the greater the semantic difference. Difference in height implies that one word meaning (or synset) is at a higher abstraction level than the other, and such difference in abstraction level should reduce the similarity value. This reduction should be dependent on the level at which the two roles are found. The effect the difference in abstraction level should have is controlled by a constant k. Experiments should decide its value, i.e., to which extent one should emphasize the difference in levels. Equation 5.7 can be changed to:

$$\begin{aligned}
&\text{sim}(w_t, w_b) \\
&\quad = 1 - \frac{(\text{lev}(w_b) - \text{lev}(w_p)) + (\text{lev}(w_t) - \text{lev}(w_p))}{\text{lev}(w_t) + \text{lev}(w_b)} - k\frac{|\text{lev}(w_t) - \text{lev}(w_b)|}{\text{lev}(w_b) + \text{lev}(w_t)} \\
&\quad = \frac{2 * \text{lev}(w_p) - k * |\text{lev}(w_t) - \text{lev}(w_b)|}{\text{lev}(w_t) + \text{lev}(w_b)} \qquad (5.8)
\end{aligned}$$

The semantic similarity between role models

The total semantic similarity sim(T, B) between two role models, as used during retrieval, can then be estimated as equation 5.9. sim(T, B) thus represents an upper bound on the analogical similarity between the target model and a base model:

$$\text{sim}(T, B) = \frac{\sum_{j=1}^{m} max_{i=1}^{n} \text{sim}(w_i^b, w_j^t)}{m} \qquad (5.9)$$

where m is the number of roles in the target, and n is the number of roles in the base. One of the assumptions I made concerning the similarity model above is that what word meaning is chosen does not matter, as long as the same synsetID is used. All synonyms would give the same value as the exact same word meaning. This is, in many cases, not correct; see for example 4.1.

No claims can be made concerning exactness and generality of the semantic similarity based on computations. My goal is pragmatic, though: To find an approximation yielding the best possible results. When the similarity model is being used during retrieval, the value that is calculated can be considered an upper bound of the semantic similarity. The imprecision in calculation due to the fact that the same two different roles in the base model may be mapped to the same role in the target model and that there is no guarantee that the chosen similarity will be used during mapping, causes me to conclude that if the similarity model gives results that are sufficiently good, there is no reason to try to get a model that may give more precise semantic similarity measures.

5.3.3 Adjustment for Possibility of Error in Sense Selection

Sometimes, when the software developer is creating a new role model, it may be difficult to choose among different meanings of the same word. The problem is how to handle a situation like the following: The software developer has selected a sense of a word for a role that does not have the correct meaning. With one of the two semantic similarity models suggested above, this will, in some cases, give a similarity of 0, if this sense belongs to a different WordNet hierarchy than the roles it is compared to. To compensate for this, and to try to prevent a model from getting a too low score to be selected for a mapping, a small similarity value could be given in such cases. Let sim(t, b) = sim(w_t, w_b) mean the similarity value that has been calculated between two roles. Let wf$_t$ and wf$_b$ mean the word forms that name the target and base roles, respectively. The constant λ represents a value that is given to the similarity value in cases where the same word form has been selected for them, but where the similarity value is 0, i.e., the meanings belong to different WordNet hierarchies. This is shown in equation 5.10.

$$\text{sim}(t,b) = \begin{cases} \lambda & \text{if } (\text{sim}(w_b, w_t) = 0) \text{ and } (\text{wf}_t = \text{wf}_b), \\ \text{sim}(w_t, w_b) & \text{if } (\text{sim}(w_t, w_b) \neq 0) \\ 0 & otherwise \end{cases} \tag{5.10}$$

Notice that it is only a question of a small adjustment, which is performed under the particular condition mentioned above. Often this adjustment will not have any influence on the upper limit of the semantic similarity between a role in the target model and the roles in the base model. A value of 0 is not likely to be the result of comparing one role in a target model to all the roles in a base model during retrieval. Remember that the best value is always chosen. However, this solution might instead have some effect during mapping.

5.3.4 Test of Semantic Similarity Models

The two semantic similarity models described in section 5.3 were tested using spreadsheets. The WordNet hierarchies are seen as a set of trees, where each *node* represents a synonym set, and where the role names being compared can be located to such a node. Each of the similarity models are used to compare nodes at different levels in such a tree.

Information about the levels of two nodes being compared, lev(w_t) and lev(w_b), and that of their closest, common ancestor, lev(w_p), is used. If w_t and w_b belong to different trees, there is no similarity between them, i.e., sim$(w_t, w_b) = 0$. If they belong to the same *synonym set*, sim$(w_t, w_b) = 1$. In all other situations sim(w_t, w_b) is between these outer bounds.

The goal of this analysis is to test how well the models spread the similarity values. A good spreading of values would be advantageous as it would permit us to distinguish between good and less good analogies. It would permit me to get an overview of the behaviour of the similarity models without having to run all the experiments. Similarity values must be within the range of 0 and 1. Both plots showing the difference in similarity value based on difference between levels of target and base node, and difference when the level of the two nodes closest, common ancestor had to be generated.

Figure 5.4 shows the similarity values for differences in level between target and base for different levels of target. When both target, base and their common ancestor is at level 1 in the same tree, the similarity value is 1. This only happens when the three role names all belong to the same synonym set. Figure 5.5 shows such a plot for the similarity model in 5.8 when $k = 0.15$.

It is important, however, to see to it that the similarity value can never be negative, so therefore k must not be too large. As the most commonly used words are generally found at level 4–6 in WordNet, the difference in level between any two words that are compared will seldom exceed 4 (if they exist in the same hierarchy), and as only very few words exist above

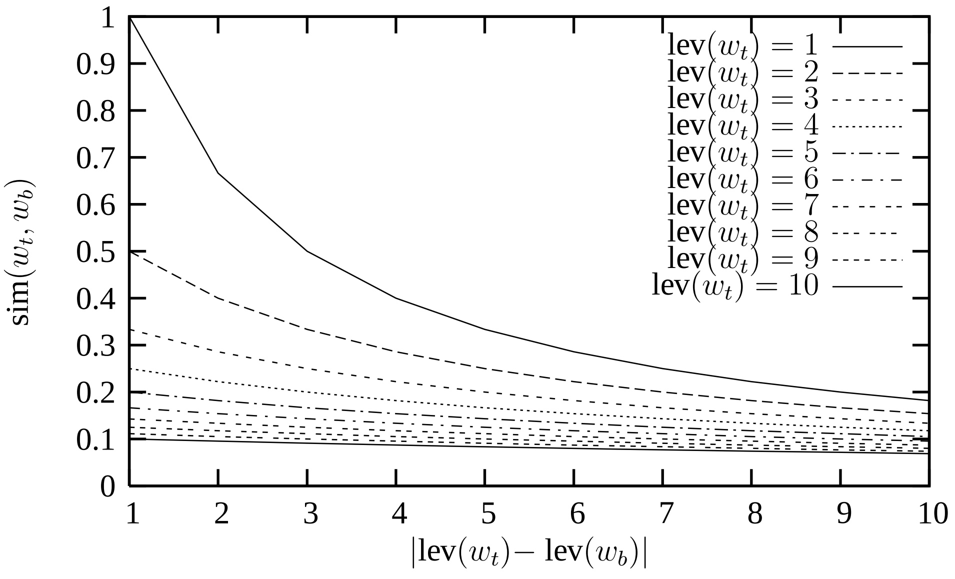


Figure 5.4: Similarity model in equation 5.7: Each plot lev$(w_t) = i$, where $1 \leq i \leq 10$, has lev$(w_p) = 1$ and increasing difference between w_t and w_b.

the level of 10, we see that we can distinguish between them using this model.

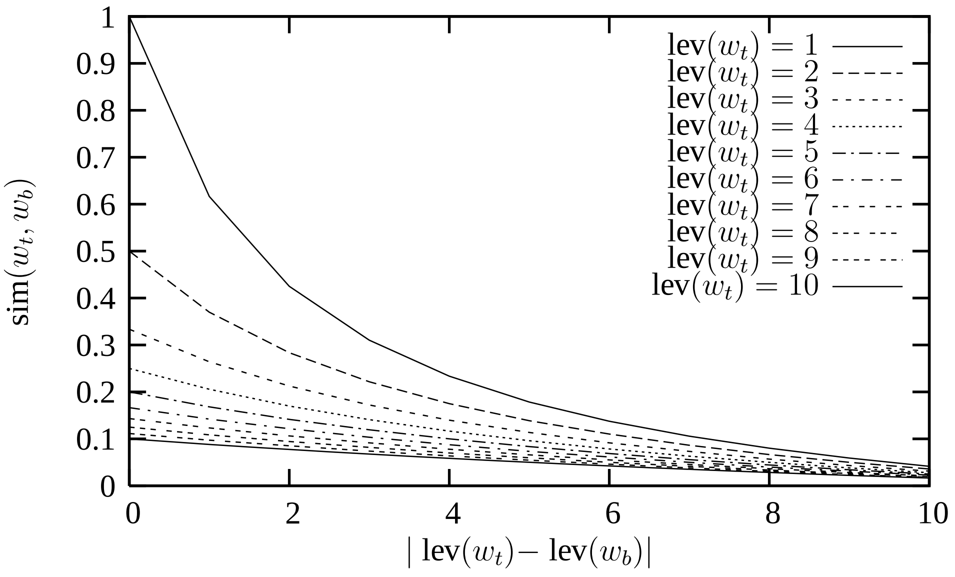


Figure 5.5: Similarity model in equation 5.8: Each plot lev$(w_t) = i$, where $1 \leq i \leq 10$, has lev$(w_p) = 1$, increasing differences between w_t and w_b and $k = 0.15$.

Comparison of the models

The preceding discussion has only shown an illustration of how the models would distinguish between word meanings during selected circumstances. The plots shown are for a selected situation where the closest common ancestor, w_p, is at level 1, and where varying levels for w_t and w_b have been selected. Table 5.3 shows a few different situations where the similarity values are calculated using the WordNet tree in figure 5.3 (a). For the model in equation 5.8 the selected values of k is also shown.

Table 5.3: Calculated similarity values between w_t and w_b in figure 5.3 (a) using the different similarity models, where $|\text{lev}(w_t) - \text{lev}(w_b)| = 1$

Sim. model	**lev**(w_p)	**lev**(w_t)	**lev**(w_b)	k	**sim**(w_t, w_b)
5.7	3	5	4		0.667
5.8	3	5	4	0.1	0.656
5.8	3	5	4	0.15	0.65

When $|\text{lev}(w_t) - \text{lev}(w_b)| = 1$ the effect of k is very little. Table 5.4 shows the effect on the similarity values when this difference is increased to 2, as is the case in the example tree in figure 5.3 (b). Here only one value of k is used in the relevant model. The difference between the two trees is only that the target has been moved one level up—to level 6.

Table 5.4: Calculated similarity values between w_t and w_b under different circumstances when $|\text{lev}(w_t) - \text{lev}(w_b)| = 2$ as in figure 5.3 (b)

Sim. model	**lev**(w_p)	**lev**(w_t)	**lev**(w_b)	k	**sim**(w_t, w_b)
5.7	3	6	4		0.6
5.8	3	6	4	0.15	0.57

5.3.5 Collocation of Terms

Reenskaug et al. (1996) suggests that roles should be given names that are as generic as possible in order to increase reuse of role models. Although this has been my goal, it may sometimes be difficult to do so because one can risk that several roles in one model end up with identical names. In this situation, new words have been added to the repository, because a more detailed vocabulary was needed. This could seem like a good solution at first, but it is easy to see that the repository could grow uncontrollably. An alternative to this approach is described in section 4.3.3. A role name could be described by more than one word form. Several noun phrases are typically a collocation of a noun and a modifier, where the modifier can be another noun or an adjective. If I introduce the possibility to code an OOram component name with a list of word meaning pointers, it seems natural that, at least for nouns, the last word should be the most significant. This means that the last word should be given more emphasis in terms of giving the total semantic similarity between two roles. In section 4.3.3, I give examples, e.g., `library assistant` and `shop assistant`. `Assistant` is the part that gives the main meaning to these terms, and the first part of them gives additional information.

Messages will often need names that are a combination of a verb and a noun, e.g., *add name*, *add phone-number* for a role *employee*. In this situation it seems intuitively natural to put more emphasis on the verb.

5.4 Example

To illustrate the use of the similarity model, the OOram role models from figure 1.4, that shows the use of the ROSA search tool displaying a selected base model, are analyzed for semantic similarity. All the stages of calculating the similarity are described. The results when comparing these models will be shown, based on the similarity algorithms presented in this chapter.

The two models in figures 4.1 and 4.2 are not so well suited as illustrations in the case as the role names with the greatest similarity here are at the same level in the hierarchy. Also, these models have the same structure. The parts of the WordNet hierarchy used in these examples are shown in figure 5.6.

5.4.1 Structural retrieval

The structure descriptions are stored in the repository when the models are first entered. For the two models involved in this example, I get the following structure descriptions.

Model	**Structure Indexes**		
	1	2	3
Wholesaler_Sub1	S 2	C 5	T 2 2
LibrarySub0	C 4		

Table 5.5: Structure indexes for the models in the example

The target model in this example has three stored structure indexes, while the base model has only one structure index. **C 4** means that the structure is a chain consisting of 4 nodes, **S**

2 is a star with two branches, **C 5** is a chain with 5 nodes, and **T 2 2** is a tree with a trunk of length two and with two branches.

There is little structural similarity between these two models, as the star is considered the most important in the more complex model. The base model is ranked as number 124 among all the models in the repository. The calculated belief is 0.33059.

5.4.2 Semantic retrieval

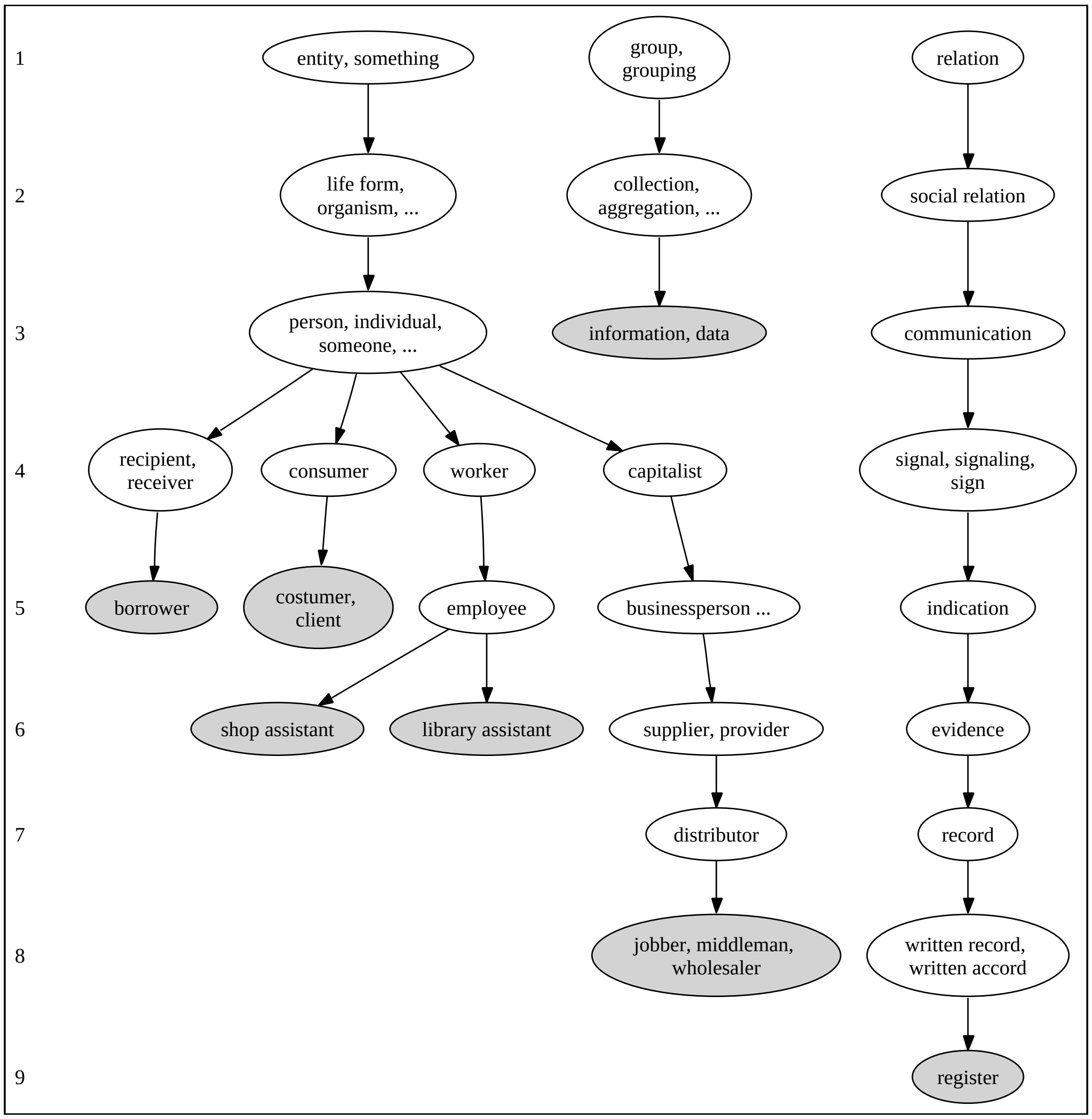


Figure 5.6: Excerpts of WordNet hierarchies for word meanings representing the role names used in the example

The nodes with a gray background in figure 5.6 represent synsets containing a a word meaning that has been used to name a role in one of the role models in the example. The levels of the nodes are marked at the left of the figure. Two of these role names are used in both the models being compared, i.e., *information* and *register*. The role *library assistant* in model LibrarySub0 was created by inserting a new word form (and therefore also a word meaning and synonym set) into the term base.

The target model—Wholesaler_sub1 in figure 1.4—contains five roles, while the wholesaler model in figure 4.2 only contains four roles. Actually, a mapping like the one shown in figure 1.4 could be used to identify flaws in modelling, e.g., the inclusion of the role *wholesaler* may be superfluous, as it represents the domain and probably not an object in the application.

Role model	**Role name**	**tree**	**level**
Wholesaler_Sub1	customer	entity	5
”	wholesaler	entity	8
”	shop assistant	entity	6
”	information	group	3
”	register	relation	9
LibrarySub0	borrower	entity	5
”	library assistant	entity	6
”	information	group	3
”	register	relation	9

Table 5.6: The role models compared with information on the role names used, the WordNet hierarhies where the word meanings of the names belong, and the synset levels in these hierarchies.

The table shows the two role models with their role names, and information about what WordNet hierarchy the word meaning of each role's name is found and at what levels the synsets can be localized. The hierarchies are named after a word contained in its root synset. The important thing here is that it is possible to show what role names are taken from what WordNet hierarchies.

When the role names are compared, we get the highest similarity values for each of the roles as shown in table 5.7. The role names *information* and *register* are actually taken from the same synonym sets in the role models, giving a semantic similarity value oft 1. Two of the roles in the target model, *customer* and *wholesaler*, get their highest semantic similarity values when compared to *borrower*. During the mapping phase only one of the target roles can actually be mapped to that base role, but at the moment I only try to find the best possible solution.

$role_{target}$	$role_{base}$	Sim. model in equation 5.7	Sim. model in equation 5.8
customer	borrower	0.6	0.6
wholesaler	borrower	0.462	0.427
shop assistant	library assistant	0.833	0.833
information	information	1	1
register	register	1	1
Average similarity		0.619	0.772

Table 5.7: The highest semantic similarity values found for each role in the target model when it is compared to all roles in the base model according to the different similarity models.

Table 5.7 shows the role names for both target and base roles in the two first columns. As can be seen, when *wholesaler* is compared with the role names in the base model, both similarity models give the best similarity value when compared to *borrower*. For the model in equation 5.7 the distance between this similarity value and the value between *wholesaler* and *library assistant* is closer due to the correction for difference in levels.

In this example $k = 0.15$. If this value had been increased, the similarity between *wholesaler* and *library assistant* would have become relatively higher. However, even when $k = 0.35$, the role name with the highest similarity value would not have changed. When comparing nodes where the difference in levels is greater, it must be ensured that the values are positive. If $w_p = 1$, and $k = 0.3$, and a difference in levels between target and base role is 7, the similarity value would become negative. To prevent this under all circumstances, $k = 0.15$ even though this will not influence the similarity value to the extent that we would have wanted.

Since the roles *customer* and *borrower* are at the same level in the hierarchy, there is no effect in the adjustment of similarity values based on difference in level, so equation 5.7 and equation 5.8 give the same similarity values. The same also goes for *shop assistant* and *library assistant*.

From the average similarity value a belief value for the semantic similarity value is calculated just like before. The belief value when the similarity model 2 has been used is 0.7448.

The combined belief value for the model *Library_sub0* is then 0.55403. This ranks this model as number 74 among all the base models after retrieval. The results of this mapping is shown in table 5.8.

wholesaler	borrower	0.427
shop assistant	library assistant	0.833
register	register	1
information	information	1
Tot. mapping value		0.652

Table 5.8: The final mappings between the two models using the semantic similarity model in 5.8

One could argue that borrower should have been mapped to customer, and that wholesaler is left out. The structure in the model does not suggest this, however, as both customer and wholesaler has a port to shop assistant. If a user sees this mapping, though, he could maybe reflect on the usefulness of adding the role wholesaler to the model in the first place. It is a typical error among unexperience modelers to include a role (or Entity type) for the application domain, while that role is actually taken care of of the whole application that is generated. This indicates that a tool such as this could also be used as a learning tool for the inexperienced developer that can learn from the models that exist in a reuse repository.

5.4.3 Similarity during mapping

When a mapping is performed between the two models in this example, for each role model a matrix is created representing the structure of the model, as shown in table 5.1 for *Library_sub0*. To represent the semantic similarity model, the matrix is created where the pairwise semantic similarity between roles in the two models are represented, as shown in table 5.2.

The genetic algorithm tries to map as many roles in the models as possible, and for each generation the semantic similarity values are used to find the mappings value.

5.5 Modifications of the Structural Similarity Model

Up until now I have only discussed what factors might influence the calculations regarding semantic similarity. There is another aspect related to the structure of the models that is important, which has not been taken into consideration in Tessem's algorithm: An experiment was performed within the ROSA project, where a group of students and faculty members were asked to identify analogous models. Each person was given three sets of models, each containing one target model and a set of base models. They were asked to give the base models scores as to how analogous they were considered to be. At the same time they were asked to comment on what had led them to the conclusion. Some of the participants said that the multiplicity that was given to the roles in the different models was an important factor when considering analogies. In the algorithm developed by Tessem, a role either knows, or does not know, about any other role.

Note that this problem is only relevant during the mapping phase. The algorithm uses a matrix for each role model, where a location represents a potential connection between the two roles in the specified direction. In OOram role models the connection that permits a role to send messages to the role on the other side of the path, is marked with a port. A single port denotes that the role knows about one role on the far side. A double port denotes that the role knows about several roles on the far side of the path. To allow for the distinction between these situations in the current setting, I may modify the algorithm to use the following values as elements in the matrixes.

0 – means that the role does not know about the role on the far side
1 – means that the role knows about one role on the far side
2 – means that the role knows about several roles on the far side

A matrix is created for each of the target and base models. This matrix shows the paths that exists between the roles in the model. A matrix for the model in 4.1 is shown in table 5.9.

	borrower	library assistant	register	information
borrower	0	1	0	0
library assistant	2	0	1	0
register	0	0	0	2
information	0	0	0	0

Table 5.9: Matrix graph representation for the library sub-model when MANY ports are taken into consideration

This information is important because it may tell us something more about the roles, e.g., a role that knows about several roles on the other side may be a collection of these roles, or it may be a role that typically handles a lot of instances of that type or role. Other roles are always passive, typically objects representing an entity in the real world about which the system stores information. This change of the structural similarity model needs to be checked.

5.6 Discussion

In some previous attempts of using semantics for software reuse, it seems that the term space has been created more or less at random or based on a guess of how much work it will take to convert one component into the other. This approach suggests that each term is connected to one particular component. In Karlsson et al. (1992) the distance between any two terms is based on how much effort the modification of the component A to component B will require. The examples they use are typically code components, e.g., classes.

Other sources Gulla et al. (1997) discussing similarity measures or distance in a semantic net, indicates that distances are given by users based on a subjective understanding. The semantic net is often constructed manually of the terms that are used in the examples, perhaps after a domain analysis has been performed. If the net is created without any lexical knowledge, it is not likely that the values given will be reliable, and two people who add them will probably not give the same value for the distance between any set of two terms.

According to Gulla et al. (1997), *car* and *book* are not related semantically. They discuss the problem of semantic similarity between two models, where one is concerned with *customer rents a car*, while the other is concerned with *borrower lends a book*. In WordNet, version 1.6, if the first meaning of *car* and *book* is chosen, both words belong to synonym sets at the 8th level in the same tree[1]. They have their last common generalization at level 3. This part of the WordNet hierarchy is shown in figure 5.7.

If trying to find the similarity between these terms based on the models presented in equation 5.7, the following result is observed

$$(4 * 2)/(8 + 8) = 0.5 \tag{5.11}$$

Because the two roles meanings are at the same level in the term space, equation 5.8 will give the same result.

[1]In version 1.7 of WordNet, the first sense of car is at level 9. This would change the values in equations 5.11 but the point would still be valid.

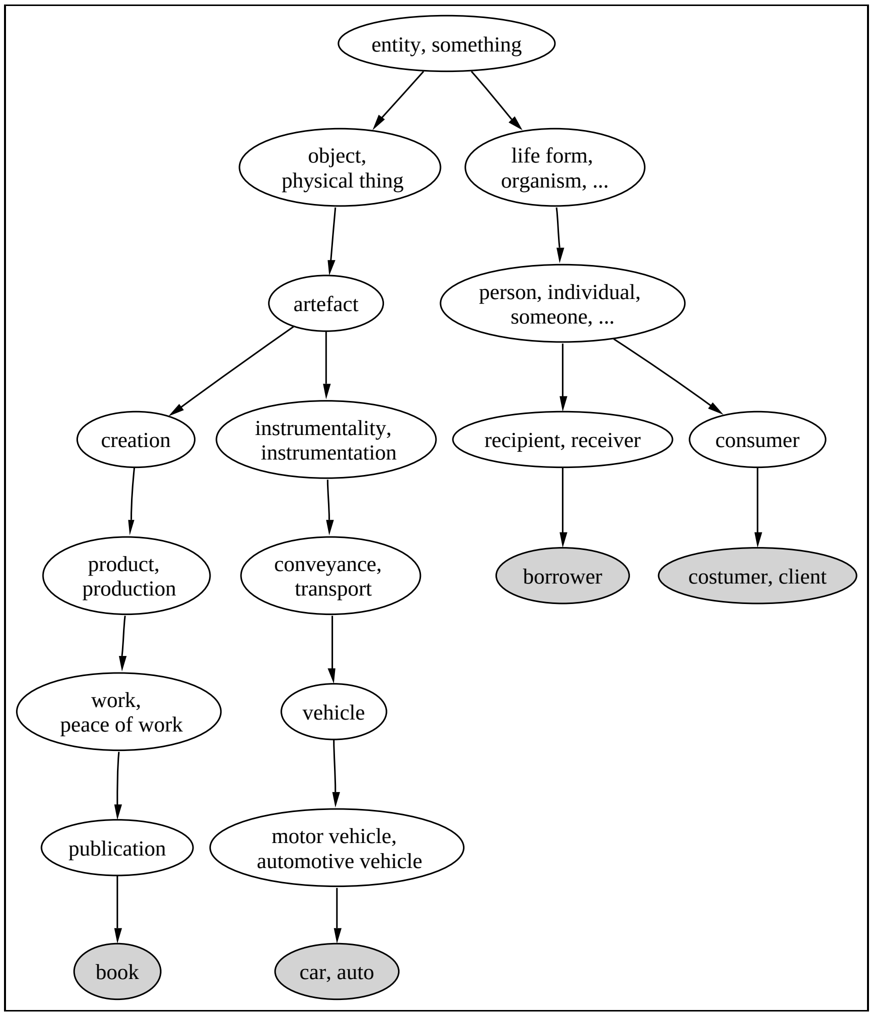


Figure 5.7: WordNet hierarchy for nouns in this example

If the words *borrower* and *customer* are used, they are also related semantically. Both are at the fifth level, and they belong to the same tree up to and including the third level.

$$(4 * 2)/(5 + 5) = 0.8 \tag{5.12}$$

From this example, it is clear that when using WordNet's database, these words are semantically similar. All words belonging to the same WordNet hierarchy have some degree of semantic similarity, and actually the $\{$*entity, something*$\}$ hierarchy contains most of the role names that have been used in the role models in my experiments.

This example shows that to be able to find similarities among models within different domains, it is necessary to have a rich lexical database with several types of relations. The conclusion of Gulla et al. (1997) stating that *car* and *book* are not semantically related is, when using WordNet, shown to be incorrect.

Chapter 6

Prototype for the ROSA Case Tool

The ROSA repository has gone through several stages. Initial work on an object-oriented repository model was done by Steine-Eriksen (1995) modelling the static object model and extended by Midttun (1998), under my supervision, with functionality for retrieval etc.

I have made considerable extensions to the model since then, particularly regarding support for the semantics of role models, semantic similarity models, and support for the mapping phase, but also support for the richer information related to the model structures. This was made possible by the use of a DTD (Document Type Definition) describing OOram role models.

In this chapter I describe the design of the most important sub-systems of the prototype for the ROSA CASE Tool. There was a need to refactor the repository based on the introduction of new tasks and new types of information. Efficiency considerations related to ObjectStore required redesign to avoid the creation of too many objects that were just pointers into the database. Issues related to the overall repository representation are described in section 6.1. One major design change is related to realizing the WordNet model. Two alternative approaches are tried, one where each OOram component is described by one term, and one where it is permitted to use a kind of modification to the assigned main name. The new organization is described in section 6.2.

In the previous prototype, the OOram role models were kept in a simple text file format. This format was difficult to extend to support the additional information needs, for example synonymy and multiplicity. To make the experimental environment more flexible, the model data are now kept in XML files between runs in order to make it possible to start from a clean database whenever needed. A Builder pattern, consisting of reader and writer classes, was developed. It permits conversion between the text file and XML file or database format, and it could easily be extended to convert from the database format and back to the XML file format. This permits for easy population of the database. This subsystem is described in section 6.4. The DTD for the OOram role models is shown in appendix C, and appendix D shows an example role model using this DTD.

The OOram part of the repository has undergone quite a few changes, and the classes of the new version are described in section 6.5. Other classes perform automatic indexing (see section 6.6) or are responsible for the analogical mapping (see section 6.7). No objects of these types are stored in the repository. I will not go into detail about the design of these classes.

6.1 The Repository Model

The ROSA repository can be considered to consist of two main parts:

1. the *OOram Model Space* with all the related information
2. the *WordNet Term Space*

The OOram Model Space consists of several collections of specific components that are interconnected, e.g., projects, OOram components, model index structures, and users. The OOram Model Space consists of **TermSpace** and connected classes modelling the WordNet database. These two main parts are interconnected by relationships.

The access to objects in an ObjectStore database, is implemented by going through a database *entry point*. Most of the access to the database goes through the OOram Model Space. Previously, ie., in the solution implemented by Midttun (1998), a new object was created every time an entry point is accessed. If an application makes frequent accesses, there will be many such entry point objects, and it may be difficult to know when such objects can be safely released. The creation and deleting of objects slow down the application, and a severe memory leak might be the result.

I therefore decided to do a redesign where a well-known design pattern (see section 2.2.2 named a **Singleton** (Gamma et al., 1995)) is used. The Singleton used in this context makes sure that all access to the database goes trough a single database entry point[1]. All database collections are thus accessed through this **Singleton** object—named **DBRoots**—from any application. After this redesign, the **DBRoots** is the only entry point to the database. Figure 6.1 shows the top level classes of the repository. Only the relationships that are related to the **DBRoots** class are shown here.

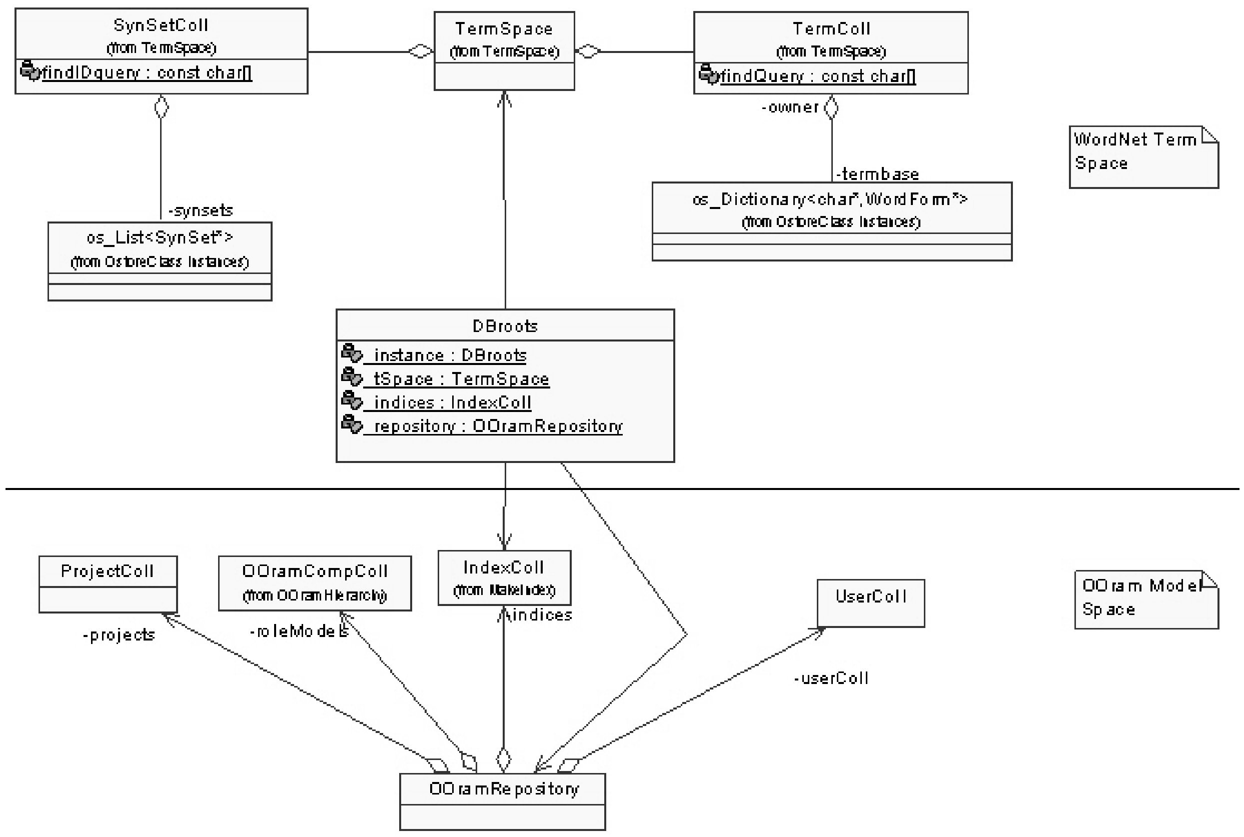


Figure 6.1: The top level repository structure

DBRoots has direct relationships to the **TermSpace**, **OOramCompColl**, and **IndexColl**. This makes it easy to both add new models and to search for analogous role models. The goal is to make it easy for the applications to access the different parts of the database in a homogeneous way, and for the cross-relationships to be easy to navigate.

[1]This may not be a viable solution if the database is used as a distributed, multi-use system.

6.2 The Structure of the Term Space

The term space in the current version of ROSA is modelled after WordNet as described in chapter 4. Two versions have been used. In both, a named OOram component has a relationship to a *word meaning* instead of to a word as before. One advantage of this approach is that I now allow for several meanings of one word. One disadvantage, however, is that the user, in situations where the selected name has several meanings, must decide what meaning to choose. To help the user pick the intended meaning, most terms in WordNet have accompanying descriptions that are also made part of the ROSA term space. A new version of the repository meant for a systems developer, should aid the user in selecting word meanings without having to select the **synsetID**s as I have done in the prototype.

6.2.1 Simple Solution

In this approach the whole name of an OOram component must exist in the term space as a single phrase. The intention of using WordNet is to free the systems developer from having to add new terms to the term space and thus to require that the developer use the existing vocabulary. The only exception to this rule should be some special terms that are naturally used within special technical domain. Figure 6.2 shows the classes involved in this new version of the term space. The descriptions of the synonym sets are not included.

WordNet contains the vocabulary of a person with English as the first language, and with a little flexibility, and maybe a little adaptation, it should be possible to name the roles and messages of the OOram models in a natural way. The design of the ROSA term space allows for navigation up and down the synset hierarchies. This should make it possible to measure similarities between two terms in the semantic net. It is possible, through the relationships between **WordMeaning** and **OOramNamedComponent** to get the particular sense of an **OOramNamedComponent**, like a role or a message, and to find the components that use a particular **WordMeaning**.

It turned out, however, that sometimes, when making a new role model, it was difficult to do this without adding new terms, and this was very cumbersome. Whenever the systems developer wants to use a name that does not exist as such in the term space, a new term must

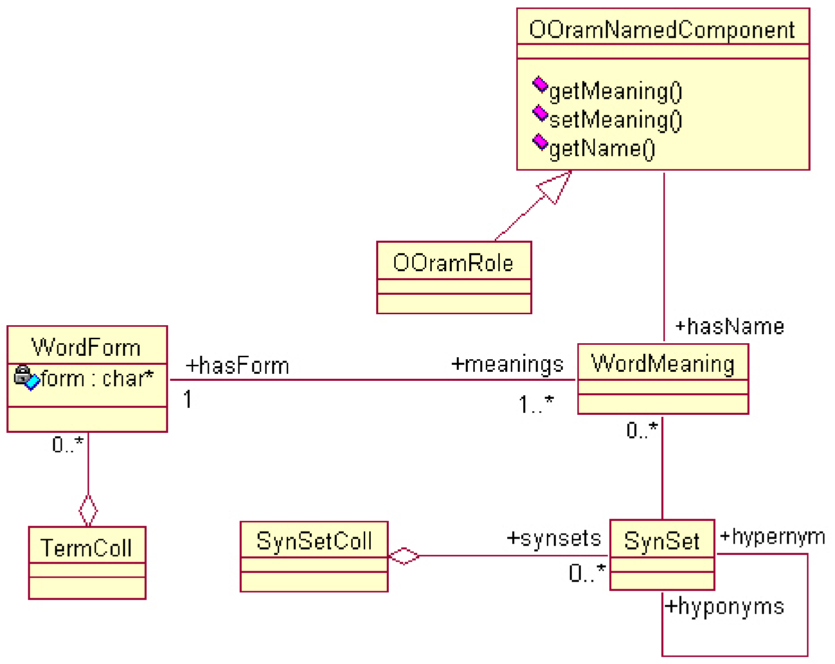


Figure 6.2: The ROSATerm Space

be added, and the system developer must decide where in the synset hierarchy the new term should be inserted. This implies decisions such as: “Should this be a new, separate synset or is the new word meaning a synonym of an existing word meaning? What is the direct hypernym to this new synset?” Such decisions are not easy, and the risk is that the quality of the term space may deteriorate over time.

6.2.2 Extension to Permit Role Names that are Collocations

Due to the need sometimes to name an OOram component by using more than one term from the term space, I looked for alternative solutions. To permit one OOram component’s name to consist of collocations of terms as described in section 4.3.3, only one change of the model in figure 6.2 was needed. A relationship was inserted between **OOramNamedComponent** and **WordForm**. Every time the system developer wants to use a name that does not exist as one term in the term base, a search is performed to see if it consists of co-occurring words, and if that is the case, the least significant word is removed, and a search is performed to see if the most significant word exists. If it does, a word meaning is selected for this word—called the *mainName*. The modified version of figure 6.2 is shown in figure 6.3.

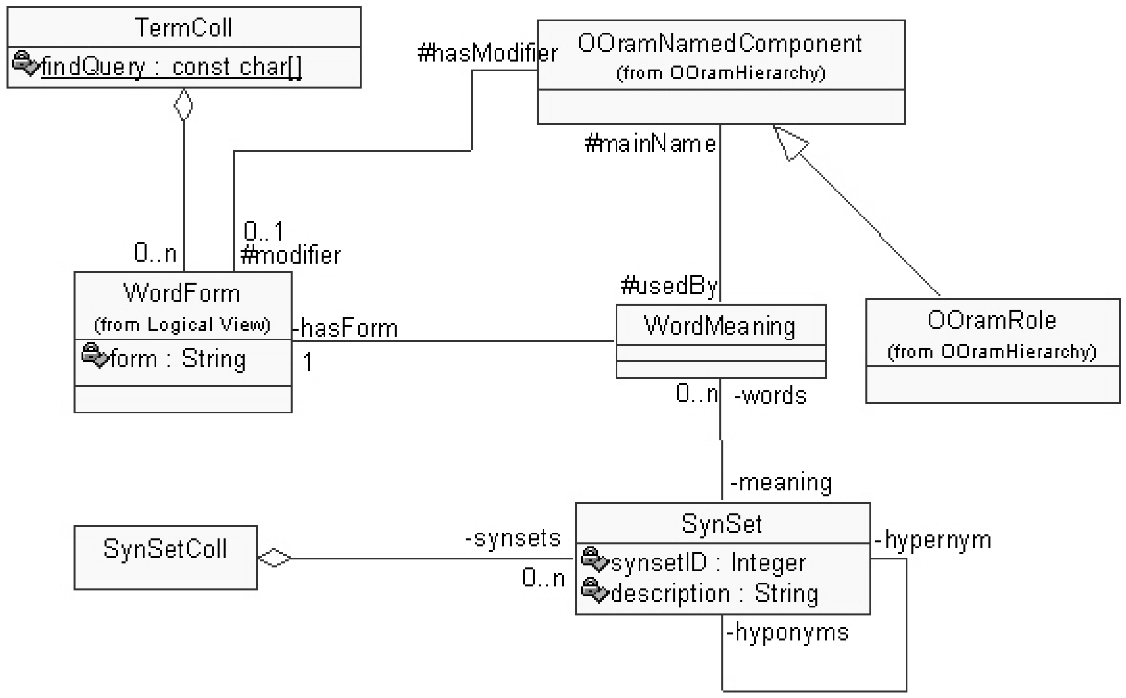


Figure 6.3: The modified ROSA Term Space

The part of the name that was removed is used as a *modifier*, and a relationship is made between the **OOramNamedComponent** and the WordForm. If the modifier does not exist either—because it in itself consists of several terms—the process is repeated. In this situation, only the last part of the modifier is used as the modifier. For nouns, this is considered to be the most significant part of the word. So far there has not been any need for this last option, so it was added merely to make the system more robust.

The database may be kept smaller by this approach, but that is not the most important reason for this change. The most important motivation is to avoid the insertion of new synsets that may lower the quality of the term space. A negative factor is that the algorithms become more complicated. The reason to go for this solution is that it frees the systems developer from having to select a word meaning for the modifier under the assumption that it is not likely that there will be a conflict when it comes to finding the semantic similarity between two roles.

6.3 The Creation of the Term Space

As mentioned before, the term space is created based on a Prolog version of the WordNet database. I currently use three of the Prolog files that are used to create the synsets, the relationships between the synsets, and the glosses. The ROSA version currently contains 116402 noun word meanings including a few that were not part of the original WordNet lexicon. The verbs and glosses have also been added, but they haven't really been used. The classes responsible for reading the Prolog files and construct the term base, are shown in figure 6.4.

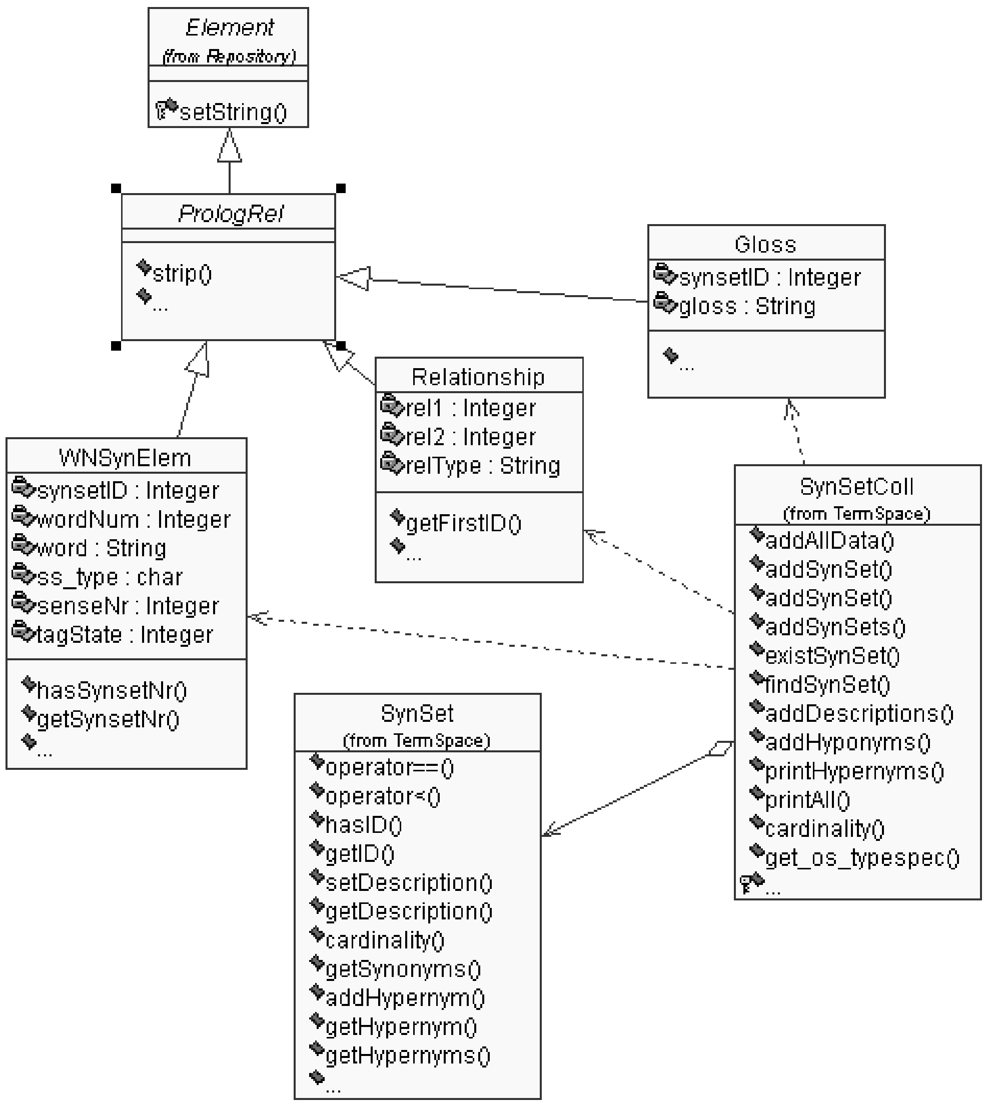


Figure 6.4: Term Space Construction Classes

One Prolog file contains the word meanings of all categories. Each line represents one entry or one Prolog *predicate*. Each predicate has the following attributes

- a synsetID unique for each *synset*
- the word's sequence number, i.e., its position in the synonym set
- the word itself, e.g., the word form
- a word category, e.g., `n` for noun
- the number of this sense of the word form
- a tag_state with the value 1 if the sense number has been assigned based on frequency of use, and 0 otherwise. This attribute is not used by us.

The class **WNSynElem** is responsible for parsing this file, and a **Synset** object is created based on one or several such objects. This is made easier because the predicates are ordered based on synsetID.

Another Prolog file contains the hyponym relationships between synsets. This gives the synset number of one synset, i.e., the hyponym, and the synset number of another synset, i.e., its hypernym or generalization. The class that parses this file is **RelationShip**. These objects are thus used to create the relationships between **Synset**s.

Yet another file contains the glosses, i.e., the descriptions for synsets and possibly examples of their usage. **Gloss** is used to parse this file. Glosses are useful when a systems developer is going to select a particular meaning of a word based on its description. In all my experiments I have pre-selected the synsetID and written it into XML-files that describes the role models.

The collection **SynsetColl** is responsible for populating the term space. It has separate methods for filling the synonym sets, the hyponyms, and the glosses. It initiates both the creation of the **PrologRel** objects, the **Synset** objects, and indirectly the **WordMeaning**s and **WordForm**s. The Prolog files are read, the term space is filled, and a copy of the database with only the term content is kept. This makes it possible to start each new experiment with a blank model base, but with the term space filled. Due to its size, this part takes the longest to generate.

6.4 Reading OOram Role Models

The responsibility for creating the OOram components from different sources and converting between various formats could have been given to the OOram components themselves. This was the approach originally taken. The result may be, however, that classes become unnecessarily complex. For every time there is a need to create an object of a particular type in a different way, a new constructor is needed, and every time a new type of conversion or format is necessary, a new method is needed. The classes may grow uncontrollably and become unmanageable.

An alternative approach is to allocate this responsibility to specialized classes, so that if we want to generate the components in a new way, e.g., from a new data format, the class that is responsible for this task, can be substituted.

Figure 6.5 is an implementation of the *Builder* design pattern described in (Gamma et al., 1995). To read and create a group of related objects from the source is separated out into a special class or set of classes. Thus input and output formats can be combined in a flexible way. In ROSA, OOram role models have been read from two different formats, and there exists two different output formats. These input and output formats can now be combined in any way. It is easy to add a new Reader or Builder class should there be a need for a new format, i.e., to build a printable version of an OOram role model. Additionally, I prevent that component classes get too complex.

6.5 Model for OOram Components

The OOram Repository has been modified from the previous prototype by introducing some abstract classes, responsibilities are moved accordingly, and some redundancies have been removed. Figure 6.6 shows the OOram repository and its relationship to the term space.

The flexibility of the design is important for later extensions and modifications of the system. The introduction of the Builder subsystem has made the OOram component classes much simpler.

The DTD used, see appendix C, permits the role models to contain other role models. This has also been implemented in the classes, but this feature has not been used in any of the

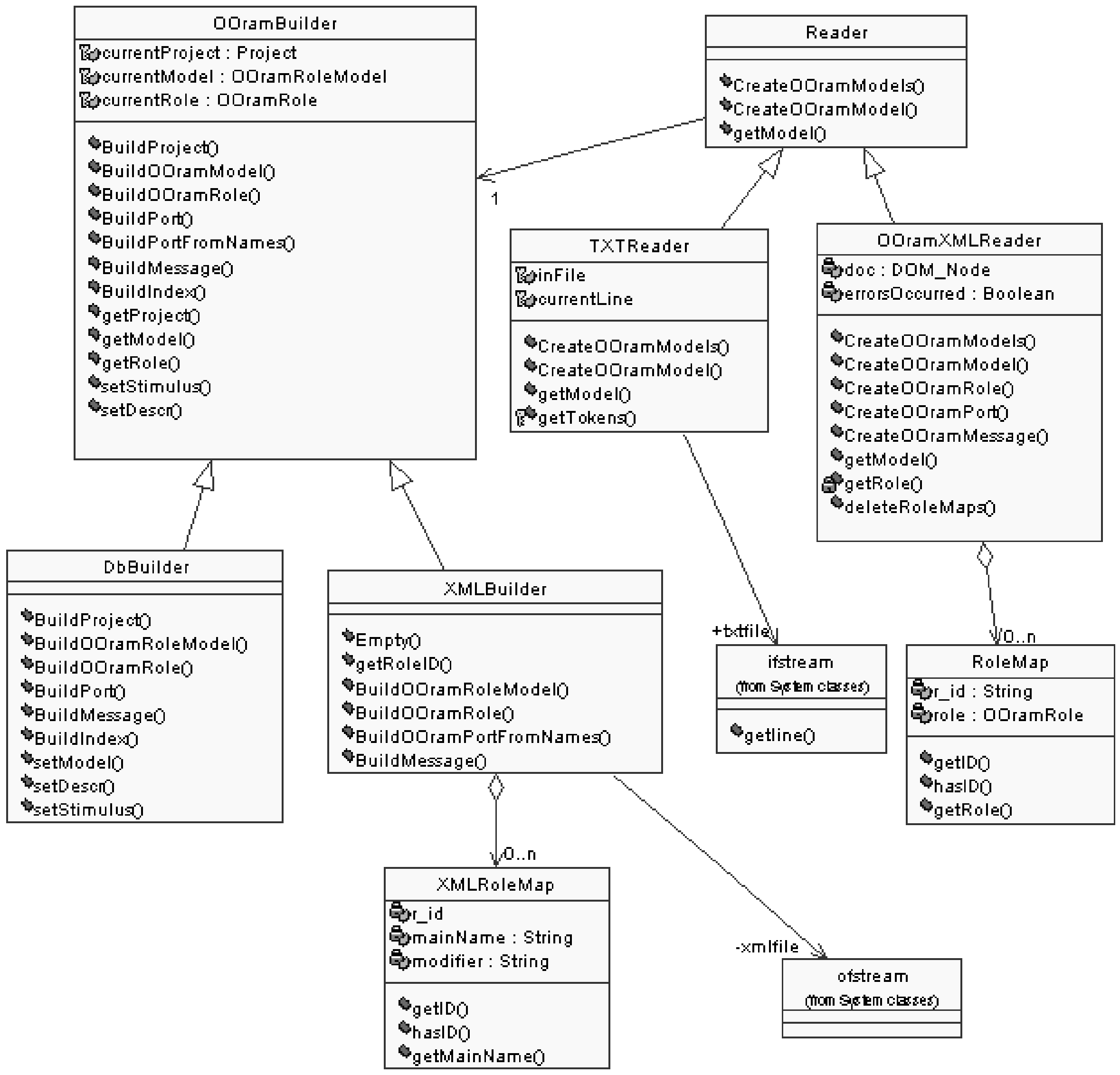


Figure 6.5: The Design for the Builder Pattern

models.

6.6 Determine Structure Descriptions

As mentioned previously, the retrieval phase of the current approach starts by searching the structure descriptions for structural similarities; see section 5.2.2. Belief functions are used to find the similarities between a target model and each of the base models in the repository. To make this search easier, a structure description for each role model is stored in a special collection in the repository when the role models are created. A subsystem has been designed

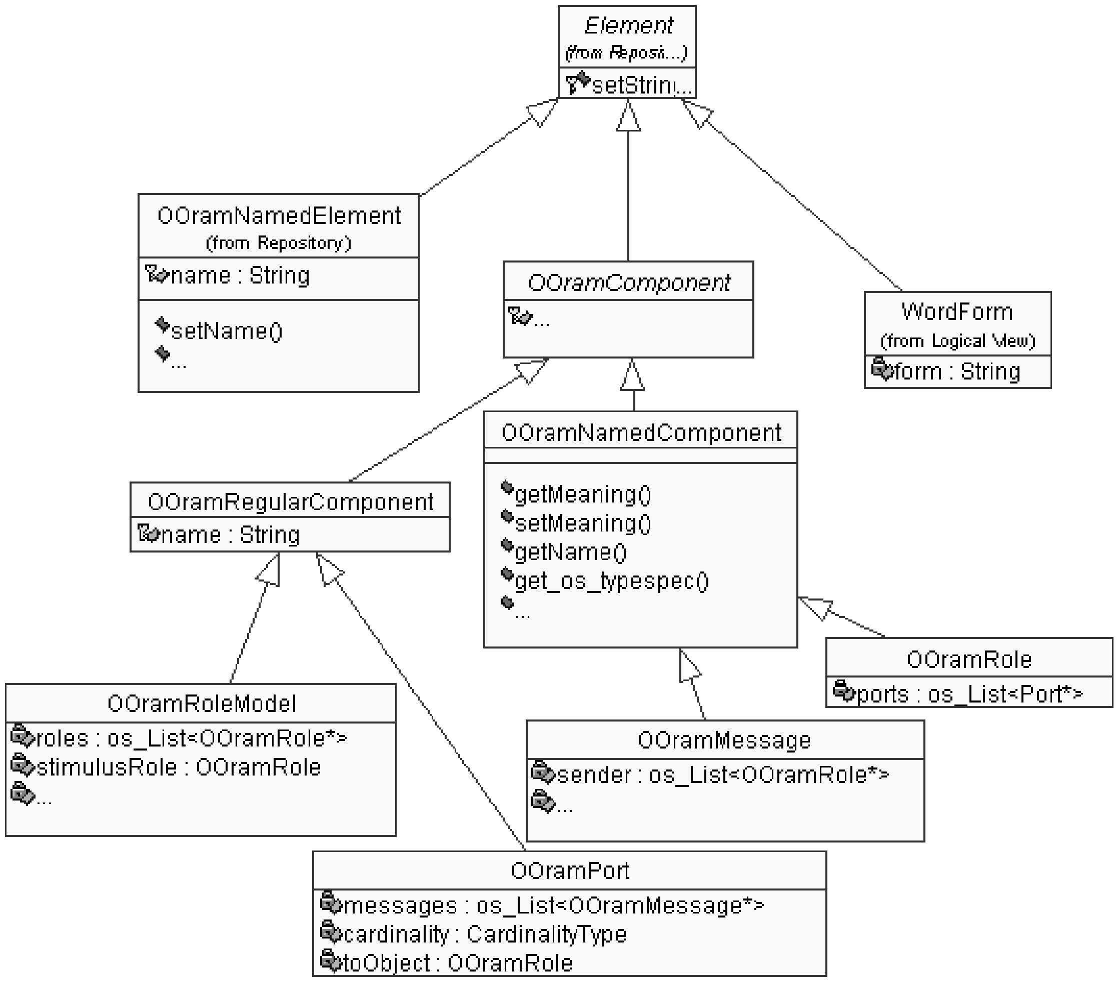


Figure 6.6: The OOram Component Hierarchy

to create these descriptions. Figure 6.7 shows the classes involved in this subsystem.

A role model is converted into a graph, and an algorithm tries to generate all possible sub-graphs that can be made with the stimulus role as its root. A value is calculated based on which structure is believed to be most dominating for the model, and up till three sub-structures are stored in the repository for each role model. Complexity in structure and size are weighted.

For each role model there can thus exist between one and three structure descriptions. The structure description contains a pointer to the role model, but the objects themselves are small. This makes the search through this collection very fast.

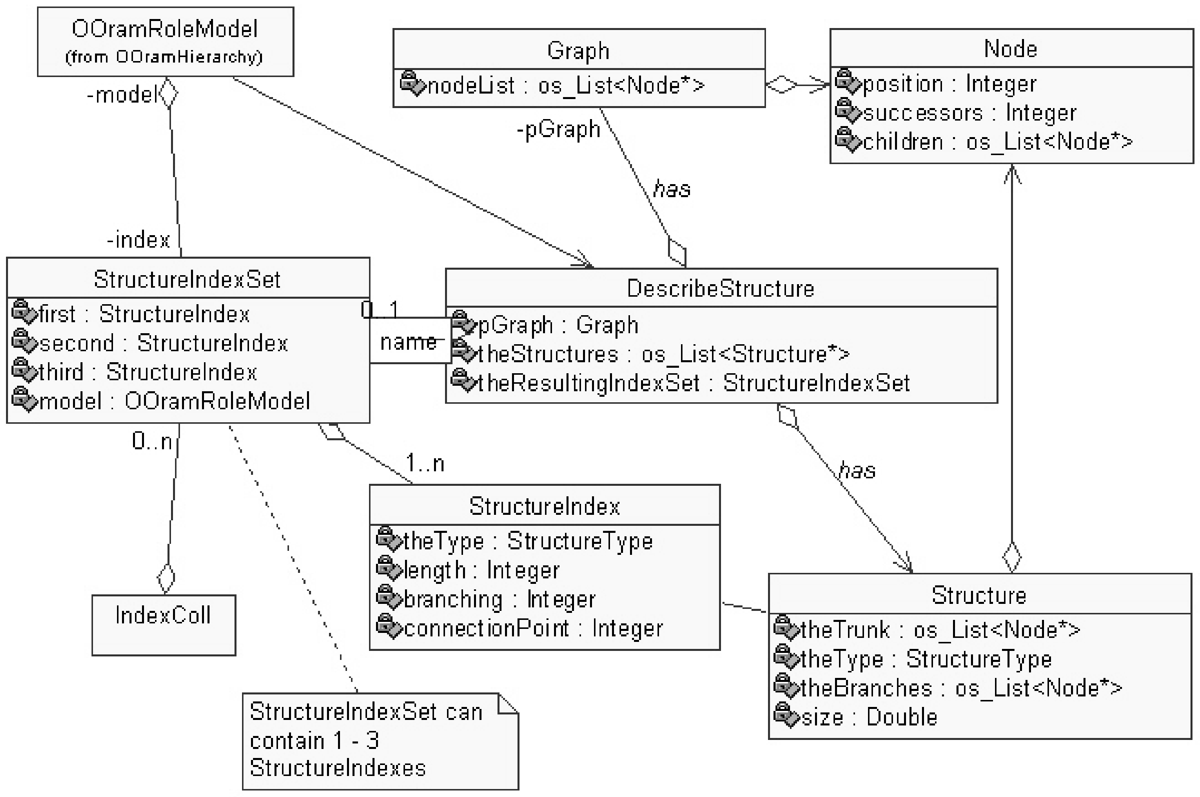


Figure 6.7: The Structure Identification Sub-system

6.7 Find Analogies

This subsystem is responsible for the mapping phase of the analogical reasoning. **Mapping-Problem** is responsible for trying to establish a mapping between the roles of two potential analogous OOram models by use of the genetic algorithm. A **MappingProblem** object is created, and, for each pair of role models that is compared, this object is initiated with references to the models that are compared.

A matrix is created for each of the target and base models to describe the existing ports within it. This matrix shows the existing paths between the roles in the model. An example matrix for the model in figure 4.1 was shown in table 5.1.

A matrix for the semantic similarity between the two roles is also created. This matrix has the role names of the target model along one axis and the role names of the base model along the other axis. The locations are filled with the semantic similarity values of the role names, see table 5.2. Here the same algorithm is used as during semantic retrieval, with the addition that if the two roles have names where the same *word form* is used, but there is no semantic similarity between them, the value is adjusted; see section 5.3.3. The adjustment factor used is 0.2.

The mapping is made by a genetic algorithm for analogical mapping. To be able to use an **Objective()** function for the genetic library, the **MappingProblem** is implemented using the **Singelton** design pattern described previously. This lets us avoid making several of the data members static. Instead, one public function, **Instance()**, is static. It is called within the **Objective()** function, and it therefore gives **Objective()** access to the necessary information about the role models. The genetic algorithm is run until a threshold has been reached, and the mapped roles are compared for semantic similarity in each generation the algorithm runs. A matrix is created that indicates the mappings between the two role models. The classes involved are shown in figure 6.8.

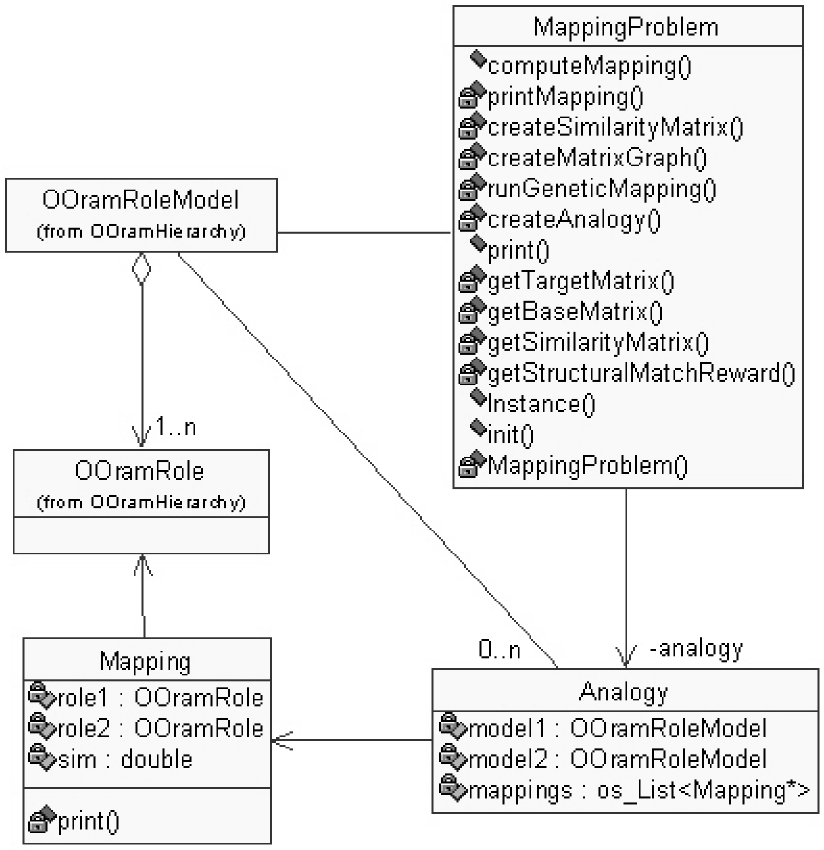


Figure 6.8: Classes Responsible for Identifying and Storing Analogies

6.8 User Interface for Analogy Search

A prototype for a user interface of the AR system is developed for illustration purposes. Figures 1.2, 1.3, and 1.4 show this at different stages of use. While this prototype shows OOram models that are created using the Taskon OOram tool and dumped to gif-files, it should ideally be incorporated with a tool graphical editor that is used to create the OOram models, and that can be used to select word meanings from the term base.

I wanted to illustrate, however, how I envision that such a tool can be. This includes how a developer can set certain parameters for the analogy search, and how the base models can be browsed to identify the good analogies. After all, some sort of human intervention is needed to see which models are the best candidates. When studying the potential analogies, the systems developer can identify other parts of the suggested analogy that can also be applicable in the new model. The analogous role model can then be investigated further, e.g., by looking at other models belonging to the same system, the design models, and maybe also its implementation. If two models are selected as the best candidates, one of them might be chosen because of the technical solution, or maybe the implementation of some roles from one model may be chosen, while the architecture of another model is identified as the best.

Chapter 7

Experiments

This chapter describes the experiment design and the 11 different experiments. The intention is to study whether the chosen approach is capable of identifying good analogies for an OOram role model under construction. If such models are identified during retrieval, we avoid performing mapping on all models in a potentially large model repository. The focus is on the retrieval and mapping phases of analogical reasoning.

Section 7.1 describes the content of the database during the experiments and the necessary fine-tuning of parameters for the genetic algorithm. In section 7.2 the experiments outlined in section 1.4 are described. An initial test establishes a *base case* (section 7.1.2). This is used as the basis for further experiments to see if the suggested AR approach can be optimized further. In section 7.4, I discuss the validity of the experiments, and finally, in section 7.3, I discuss the results of the experiments.

7.1 Preparations

This section describes the content of the repository, both term space and the OOram model space, the semantic similarity models and how they are tested using the OOram models in the repository. Finally I discuss how the genetic algorithms are fine-tuned to obtain the best possible mappings.

7.1.1 Content of the Term Space

The term space is filled with information from the Prolog version of WordNet. Excerpts from the Prolog files used are shown in appendix E. The organization of the word categories is taken from WordNet. Database and applications are created so that the descriptions and example part of the synsets— the glosses—can also be loaded. Glosses are useful when a user must select among different meanings of a word form.

Initializing the term base is a time consuming process in ROSA, but this activity is only done after modifications are made to its content. For efficiency reasons the content of the WordNet senses file is split into a set of files, one for each word category. In the experiments I only use nouns, as roles are the only OOram components in the current models that are named using the term space. I keep a copy of the database containing only the term space to be used as a starting point for each experiment.

All noun word forms with meanings from WordNet are added to the repository. Many words are not used, but I use this as an initial approach. Moreover, I have created a file with some additional terms not included in WordNet, but which I felt were needed when creating the model files. These are typically collocations.

7.1.2 Content of the Model Part of the Repository

133 OOram role models are used in the experiments. Most models are sub-models, i.e., they represent one scenario for a particular system. Many of these models have been found in books or papers that present object-oriented analysis and design, eg. in (Blaha et al., 1988; Castano and Antonellis, 1993; Gulla et al., 1997; Halpin, 1998; Kendall, 1999). In most cases the models were presented using other notations than OOram and are rewritten using OOram notation. Other models are created particularly for the ROSA project, and some of these are created to be analogous (see for example the examples discussed in section 4.2.3. It would be valuable to see if the algorithms are efficient enough when the size of the repository grows even more, but it is time consuming manually to create sufficiently realistic models in large numbers. It is vital to have a minimum number of models to evaluate the algorithms and to use statistical methods to evaluate the results. The existing number of models is adequate for this purpose.

The models are read from XML files written according to the DTD shown in appendix C. This DTD was written to cover the most essential subset of the OOram language grammar described in Reenskaug et al. (1996, p. 341). In the first experiments the DTD in appendix C.1 is used, while the DTD in appendix C.2 is used after introduction of a role name modifier in section 7.2.4. Use of XML gives the flexibility to add more detail to the role models than what was possible in the previous ROSA prototype.

While the simplest role model is a chain with one structure description, the most complex role models may be described by many structure descriptions. Since only the three most important of them are stored, not all roles will be part of a stored structure description later used during retrieval. The intention of the ROSA approach to reuse is to find analogies among sub-models, but it is also interesting to see how the system degrades as the models grow larger and more complex.

Most sub-models are simple—the smallest is a chain containing two roles—but typically they contain 3–5 roles with differing structure. On the other end of the scale there are two complex models that typically represent a high level of synthesis; see appendix A. The most complex model contains 16 roles and 36 ports.

When roles are named using word forms with more than one meaning in WordNet, a meaning is selected by identifying a **synsetID**. An example OOram role model in XML format is given in appendix D. These are the same synsetIDs that can be found in the Prolog predicates in appendix E. In a realistic system, this functionality would need support in the user interface by giving the user an option to choose from a set of meanings displayed with synonyms, descriptions, and examples. The **synsetID** would then be given implicitly.

At the beginning of each experiment, the database is filled with the *base models* for the experiment. Each experiment consists of a set of cases, each with a different *target model*. The types of target models are varied both with respect to size, structure, and semantics to get a representative set. For some models there do exist similar ones in the repository, while for others this is not the case.

7.1.3 Test of Semantic Similarity Models

Two semantic similarity models were described in section 5.3. Section 5.3.4 describes tests performed to study how these models would behave during different circumstances when two role names were compared. These tests were done using spreadsheets. It is interesting, however, to see how the similarity models behave when used on the OOram models. The similarity model in equation 5.8 was run with three different values of k: 0.1, 0.15, and 0.2. Generally, I found that this similarity model gave better results than the simple model in equation 5.7, meaning that I got higher mapping similarity for good analogies. If the goal is to separate analogous models from those that are not, it is an advantage that the mapping similarity values are spread out on the whole scale of values.

In the next section I discuss the fine-tuning of some parameters for the genetic algorithm to study how further tests with the different combinations of parameter values with the two similarity models. This shows some of the complexities involved when trying to identify the optimal values that should be used to perform the experiments with.

7.1.4 Tuning Parameters for the Genetic Algorithm

The genetic package GALib (Wall, 1996) is used in the prototype. This package is very flexible and you can set up one or several populations and predefine several parameters. The chosen implementation uses one population with the following parameters fine-tuned to improve the results. These parameters include

- *number of generations*
- *number of individuals in each generation*
- *mutation rate probability*

The goal of these tuning experiments is to identify a combination of parameters which will optimize the sum of mapping similarity values, x_n, for the n best analogies

$$x_n = \sum_{i=0}^{n} v_i \tag{7.1}$$

where v_i is the value of the ith analogy. I initially conduct a set of experiments where I modify these parameters in different combinations. To make sure that the results are sound, I check the combination of parameters using the similarity models described in section 5.3, to see what combination gives the best analogies. Due to lack of measures of what constitutes a good analogy, the *best* parameters is here taken to mean those optimizing x_n (see equation 7.1). I started the experiments using the values chosen by Tessem (1998a) as default in his experiments.

First, I ran a set of experiments that differed in population size and number of generations within the minimum and maximum ranges in table 7.1. The mutation rate had a starting value of 0.03. Several sets of tests where done with the similarity models given in equations 5.7 and 5.8. These findings led me to narrow down the tests performed later with the second similarity model.

Parameter	**min value**	**max value**
Population size	200	3000
Number of generations	250	2500
Mutation rate	0.03	0.07

Table 7.1: The outer limits of the genetic algorithm parameters tested

I ran the similarity model in equation 5.8 with population size and number of generations equal to 400, and with a mutation rate of 0.05. Thereafter an experiment was performed with size of population and number of generations equal to 500. The latter gave increased value of x_n compared to the default if n equals the number of base models in equation 7.1. After further study this was no longer obvious.

The seemingly better result, an increased x_n, was often due to an increase in mapping similarity for base models with low rank and therefore low mapping similarity. These models are uninteresting since I only look for the best analogies. The models that were intentionally created to be analogous, e.g., almost identical models, should always get the best mappings.

The goal was therefore modified to ensure that the *best* analogies always get the highest mapping similarity values possible, and to distinguish models that are highly analogous from weaker analogies. Only the 10 best analogies, i.e. $n = 10$, is therefore included when the sum in equation 7.1 is calculated. Mutation probability rates of 0.04, 0.045, 0.05, 0.06, and 0.07 were checked using the two similarity models with population/generations as previously described.

A mutation probability of 0.04 was chosen as it gave acceptable results. Using this value, a set of tests were executed where population size and number of generations varied.

Population size varied between 200 to 600, and the number of generations varied between 250 to 600.

Conclusion The chosen values for the genetic algorithm after these initial tests are: A mutation rate equal to 0.04, population equal to 400, and number of generations equal to 350. The similarity model in equation 5.8 is chosen. These vales gave the highest mapping similarity for the 10 best analogies using equation 7.1.

7.2 The Experiments

This section starts out with testing the first implementation of the AR machine in section 7.2.1. This first experiment establishes what is called the *base situation*. The following experiments are aimed at testing several modifications of this first implementation. Each of them is run with at least one *alternative situation* to be compared to the *base situation*. I call the base situation A and the alternative situation B. A table showing the sequence of experiences is shown in table 7.2.

After the *base situation* is established, 10 experiments are performed. In each experiment, the same 24 cases are run. Each case has a specific *target model*. In each case it is searched for the best analogies for this target model. The results from the two situations are compared. In each case, all models except the target model are used as base models. In this experimental situation, similarity information about all base models is stored. For each case in each situation, the following information is stored about each base model:

- the *probability* that the model is analogous to the target after the retrieval phase, i.e., the *retrieval similarity*
- the *rank* of the model according to the *retrieval similarity* (see section 5.2.2 for the belief theory and how this is converted to a analogy mapping value in equation 5.4)
- the *mapping similarity* value
- the *rank* of the model according to its *mapping similarity* value

For each case the result contains a set of base models with a calculated mapping similarity value. The 5 and 10 most promising analogies for each of the 24 test cases in the two situations are compared using

$$x_{ij}^{y} = \pi_y(a_{ij}) \tag{7.2}$$

where a_{ij} is the ith analogy in case j, and x_{ij}^{y} is the rank, π, after retrieval of the ith analogy in case j using method y. A change in x_{ij}^{y} is measured as $\delta_{ij} = x_{ij}^{B} - x_{ij}^{A}$. To study the best analogies in particular is important because this can reveal whether the results are improved by the modifications that are done to the solution.

Each experiment, except for the initial experiment testing the soundness of the genetic algorithm, see section 7.1.4, and the experiment setting the base situation, see section 7.2.1, has one out of two objectives, namely to

- improve the *rank according to retrieval similarity* in the alternative situation for potentially good analogies, ie., models with a high rank after the mapping phase, compared to the base situation
- find better mappings between the base and target models in the alternative situation, i.e., to *improve their mapping similarity values*, compared to the base situation

In experiments concerned with the first objective, the rank after retrieval, x_{ij}, of the analogies with rank i in case j for the two situations are compared. Only models that are potentially analogous, i.e., with the highest mapping similarity values, are interesting, and therefore only models with the top ranks, i.e., $i = 1 \ldots 10$, are studied. Preferably the models should be ranked in the same sequence after retrieval as they are ranked after mapping. The *ideal* is that the models are ranked the same way after retrieval and mapping. In other words that I am able to identify the best analogies after retrieval. As this is difficult to achieve, a *good* result would list the potentially best analogues in the upper part of the retrieved list, say among the top 20 or 30 ranked models after retrieval.

Experiments concerned with finding better, or more correct, mapping similarity values, i.e., better mappings between a target model and each of the base models, should compare the mapping similarity between pairs of target and base models before and after some change has been made to the algorithm that identifies analogies. The mapping similarity value is improved if more roles are structurally mapped, and if the mappings that are identified are between roles with names that are semantically close. The difference between the mapping similarity calculated in the two situations being compared is $\delta_{ij} = z_{ij}^{A} - z_{ij}^{B}$, where z_{ij}^{X} is the mapping similarity of the ith analogy in case j for situation X, and where A and B represents the two situations being compared. If the difference is positive, the mapping similarity is better.

Table 7.2 shows an overview of the experiments performed, their objectives, and the properties that are measured during the experiments. The measurement in an experiment compares a property from the situation before with the same property after the change has been made. These experiments are described in the following sections.

Experiment	**Objective**	**Measurement**
Test similarity models on real OOram models	Establish base situation	Rank after retrieval of best analogies
Map target model to itself	Test the soundness of the genetic algorithm	Sequence of target models
Compare two identical runs	Test the stability of genetic algorithm	Diff. between mapping similarity
The importance of modifiers	Reduce need to expand term space	Diff. between value after retrieval
Distinguish between one and many ports	Find better analogies	Diff. between mapping similarity values
Adjust for error in sense selection	Improve pos. of best analogies after retrieval	Rank after retrieval of best analogies
Increase weight on struct. sim. during retr.	Improve pos. of best analogies after retrieval	Rank after retrieval of best analogies
Increase weight on sem. sim. during retr.	Improve pos. of best analogies after retrieval	Rank after retrieval of best analogies
Punish large base models during sem. retr.	Improve pos. of best analogies after retrieval	Rank after retrieval of best analogies
Remove synth. models from repository	Improve pos. of best analogies after retrieval	Rank after retrieval of best analogies
Use fitness function to sort analogies	Improve sorting order of models after retrieval	Rank after retrieval of best analogies

Table 7.2: Overview of the experiments that have been performed

If not otherwise stated, all experiments are run with the parameters for the genetic algorithm and the similarity model listed in table 7.3 below.

Parameter	Value
Population size	400
No of generations	350
Mutation rate	0.04
Similarity model	Equation 5.8

Table 7.3: Chosen parameters for genetic algorithm and semantic similarity model

7.2.1 Test Similarity Models on Real OOram Models

The similarity models was originally checked on spreadsheet data, see section 5.3.4, and there was a need to establish a *base situation* using the OOram models in the repository. The parameters in table 7.3 where used. I want to find out how many role models must be transferred to the mapping phase to be reasonably sure that the best analogy is among these models for all the 24 test cases in this initial set up.

Results Figure 7.1 illustrates how many models must be transferred to the mapping phase for the best analogy, i.e., $i = 1$, to be found in all the cases. The x-axis represents the number of base models returned from the retrieval phase, while the y-axis represents the fraction of the best analogies that are identified. In 15 of 24 cases, the best analogy is found in position 1 after retrieval.

In the worst case the best analogy was ranked number 67 after retrieval, i.e. for the best analogy to be found in all these 24 cases, 67 models must be transferred to the mapping phase. If, in each case, the 26 top ranked models after retrieval are passed on to the mapping phase, I will be able to identify the best analogy in 21 of the 24 cases.

For the worst case, in which the best analogy was ranked number 67 after retrieval, the base model scored low on structural similarity as it was ranked number 123 after structural retrieval. If the two models, the target and base model in this situation, are analyzed further, it shows that the reason for the low structural similarity is that this base model has only one structure description index, i.e., a sequence, while the target model has a tree and a sequence. This base model scores high on semantic similarity, rank 6, i.e., if semantics had been given more weight, this base model would have come higher on the list after retrieval. Structural

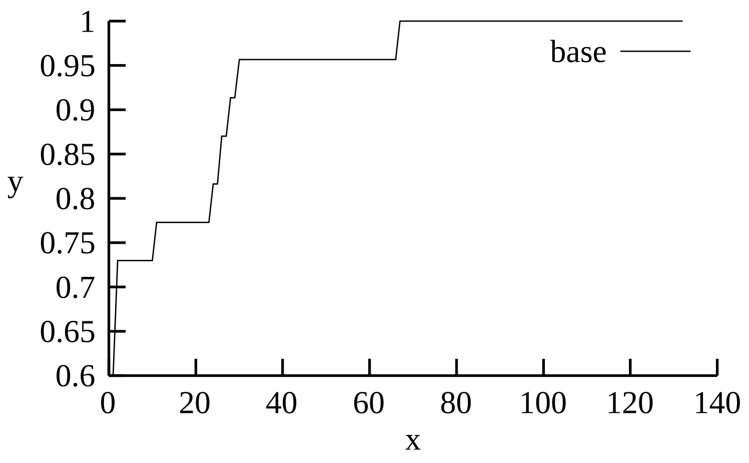


Figure 7.1: Fraction of best analogies found pr. models mapped

and semantic similarity are given equal weight and this fact accounts for the failure to identify this particular base model after retrieval.

Several cases have a target model where at least one base model is semantically similar. In some situations there are base models that are intentionally created as variants of the target model, or vice versa, and these are both semantically and structurally similar. The best models are always found in these cases. In the cases where analogous models had been created, these where also identified, even though the semantic similarity is lower in these cases. Some of these models have been used as target models (see table F.1. An example of this is shown in section 7.3. The goal of analogical reasoning is to be able to find the best analogies across application domains, so there must not be given too much weight to semantic similarity. Section 7.2.7 and 7.2.8 describe experiments where the weights on structure and semantics are varied during retrieval.

Conclusion The intention of this first experiment was to establish a base situation that can be used for comparison in later experiments. I want to see which of the alternatives give the best analogy, given that a good analogy is one with a high mapping similarity. As an initial attempt the algorithms give some indication of a viable approach. For all the best analogies to be identified during mapping, I had to pass on 67 models to the mapping phase, i.e., about 50 % of the models in my experiments. However, if only the 26 highest ranked models after retrieval are passed on, the best analogy is found in 21 of the 24 cases.

7.2.2 Map a Target Model onto Itself

One basic requirement for an AR machine is that a model should always be found analogous to itself. To test the soundness of the analogy model in this respect, an experiment was performed where the target model itself was included in the collection of base models.

Results In all of the 24 cases run with different target models the best analogy was the target model itself. In all but one case, this mapping had the value of 1, i.e. all roles were mapped onto itself. In the last case the mapping was lower because two of the roles in the base model were mapped to another role than itself. This situation can be explained by the fact that a genetic algorithm selects mutations at random, and so a local optimum was found.

Conclusion If the target model is included in the collection of base models, the best mapping is almost always between the target model and itself.
In the remaining experiments, the target model is excluded from the collection of base models, and the set of base models consists of 132 models.

7.2.3 Compare Two Identical Runs

When using genetic algorithms, there is no guarantee that the same results are produced each time the program is run with the same target model. In fine-tuning the genetic algorithm, the goal was to identify the best possible mappings. The parameters varied were the size of the population, number of generations, and mutation rate. To be able to compare different situations in an experiment, it is an advantage to know how much the results vary in two identical situations. Such an experiment was performed and the results were analyzed. The difference in mapping similarity $\delta_{ij} = z_{ij}^A - z_{ij}^B$, between the two situations was measured for all the mappings.

Results In 24 cases, with a total of 3165 mappings, only 5 mappings differed in value when the exact same set of tests were run twice. This amounts to 0.158 % of the mappings. The difference in mapping similarity in these 5 cases was 0.045, with a maximum of 0.111. In this last case, this difference did not result in a difference in rank after mapping.

Conclusion Since less than 0.2 % of the mappings vary between two identical runs, I can conclude that variations between two runs that differ in some respect, are not due to random variations in positions or mapping similarity value when the same parameters are chosen.

7.2.4 The Importance of Modifiers

In some early experiments, new terms were added to the term base every time a model could not naturally be described by terms contained therein. This situation could occur if two roles in a model have names that are specializations of a common term, and the specializations do not exist in WordNet. Adding a new word meaning usually leads to the insertion of a new synset as well. The new synset must be linked to an existing synset in the term base as a specialization. This process is error prone due to the risk of adding a new synset at a semantically wrong place in the hierarchy. Even worse, because neither I nor any typical practicing software engineer have experience in creating a lexicon, there is a risk of reducing the quality of the term base.

To cope with this situation, a modifier is introduced, as described in section 6.2.2. The introduction of a modifier allows a role to be named using two terms, and the second term, if used, is called the *modifier*. All roles with names not already in the term space get a modifier. The modified DTD for the OOram models is shown in appendix C.2.

To illustrate the introduction of modifiers, the *Library assistant* role in **Library** is changed to `rName` = *assistant* and `mName` = *library*. The role model **Library** is compared to

- itself,
- a model **Library_a**, where the same change to this role is made, but where another role has been removed,
- a smaller model, **Library_sub**, that uses the original word form *library assistant*, and thus no *modifier*.

Results When compared to itself, **Library** gets $\text{sim}_{sem} = 1$ on all roles. The roles *library assistant* has the closest common ancestor at level 6 in the hierarchy. When compared to **Library_a**, all roles, except the missing one, has $\text{sim}_{sem} = 1$. Total semantic similarity in this case is 0.944. When compared to **Library_sub**—with only 5 roles—the two *library assistant* roles get a similarity equal to 0.667. They have the closest common ancestor at level 4. Figure 7.2 shows part ot the hierarchy that involves these terms. The modifier *library* is marked with a dotted arch to distinguish it from specializations. The reason the term *library assistant* is a specialization of *employee* rather than of *assistant* is that a synset containing *shop assistant* was already linked in this way in WordNet.

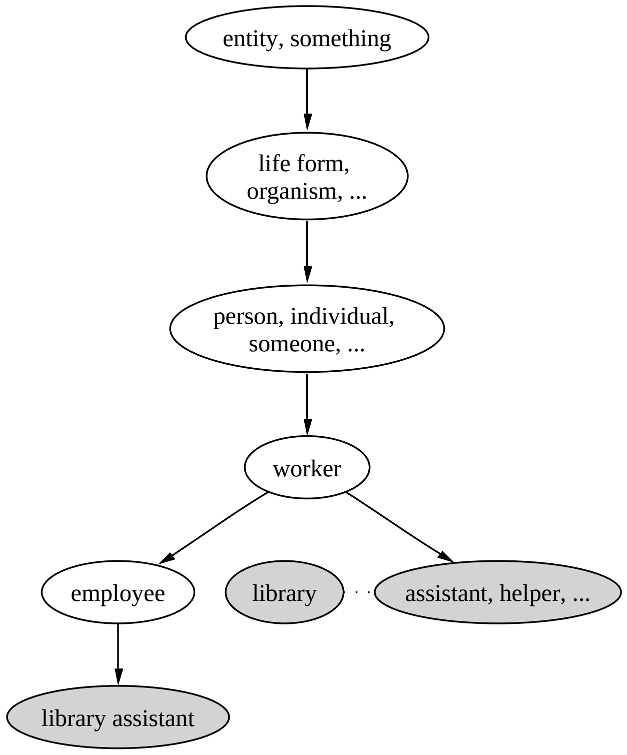


Figure 7.2: Term space hierarchy related to *library assistant*

Conclusion If two roles, named by the same word meaning, are compared, $Sim_{sem} = 1$. The same is true if both the same word meaning and modifier are used in both cases. If, on the other hand, the role name is the same, but one role is named using a modifier while the other role is named by adding a new term to the hierarchy, they do not get the maximum semantic similarity. Such a situation can be avoided by notifying the user so he could fix it. If he wants to add a new term to the term space, he should be asked to use the first approach instead. This will ensure that the term space develops in a more controlled way.

So, if a role name is written using a modifier in all situations, the similarity between two identical role names will be 1, and I therefore introduce this in the repository. It is easy to enforce this by not permitting the introduction of new terms in the term space. In the role models used in these experiments, after this change, there are 31 uses of a modifier.

7.2.5 Distinguish Between One and Many Ports in OOram

In OOram role models, a role can know about many, one, or no occurrences of the role at the other end of a path connecting the two roles; see appendix A. In the experiments described thus far, structural similarity during mapping is found based on whether the role knows about the role at the other end or not, i.e., there has been no distinction between situations where there has been a ONE or MANY port.

Now the code is modified to give different results depending on the multiplicity of the ports. I want to analyze whether the addition of such information helps to identify better analogies, i.e., with higher values. The original, or base, situation is called situation A and the alternative situation B.

Hypotheses The null hypothesis, H_0, states that there will be no improvement in mapping similarities based on the additional multiplicity information. The alternative hypothesis, H_1, states that there will be an improvement when such information is taken into account.

A change in mapping similarity is measured as the difference between mapping similarity for pairs of target and base models, a_{ij}, for the two situations (A and B). The difference is measured as $\delta_{ij} = z_{ij}^B - z_{ij}^A$.

Formally the hypotheses can be set up as:

H_0**:** median $(\delta) = 0$
H_1**:** median $(\delta) > 0$

Analysis

In this experiment, I want to find out what happens to the mapping similarity values when multiplicity is taken into consideration. In some cases, the use of multiplicity information may result in mappings with lower values, but this may still be a more correct mapping. It is without importance if these models do not show up among the top ranked analogies. For each target model, j, the base models are sorted alphabetically. In this way the analogy values of the same base model can be compared for situations A and B.

To analyze the calculated differences, a suitable statistical method must be identified. Using probability plots, the data do not show a normal distribution, so a non-parameterized method is used. The values of differences can be ranked, so the *Wilcoxon signed rank test* (Wilcoxon and Wilcox, 1964) is used. The significance level, α, is set to 0.05.

H_0 is first tested using data from all mappings. The data set is then reduced several times by incrementally removing weaker analogies. I want to find out if, above some level, H_0 can be rejected. Since the same mappings are not the best in both situations, I keep the best x mappings from both situations. Table 7.4 shows the result for this experiment.

k best mappings	N	N for Test	P	Estimated Median
$k = 132$	3165	1430	1.000	0
$k < 100$	2612	1261	1.000	0
$k < 80$	2157	1065	1.000	0
$k < 60$	1636	822	0.997	0
$k < 40$	1096	536	0.947	0
$k < 20$	1096	536	0.947	0
$k < 15$	418	198	0.117	0
$k = 10$	278	120	0.030	0
$k < 10$	250	107	0.034	0
$k < 5$	121	54	0.052	0
$k < 4$	92	38	0.025	0
$k < 3$	62	24	0.042	0

Table 7.4: Analysis of using multiplicity information with the Wilcoxon test

Results In table 7.4, *N* represents the total number of data points, while *N for test* represents the number of data points that differ from zero. P is the probability of getting the results in table 7.4 if run again, while k represents the max rank of best analogies that are kept from both situation A and B. The models that have the highest similarity mappings may vary between the two situations.

Conclusion

From table 7.4 it can be seen that when considering the 2–10 best models in either A or B, the probability P is within the significance level. It is therefore possible to conclude that when looking at the 2–10 best analogies in both situations, H_0, stating that the analogy values will not be better, can be rejected. In future experiments, I therefore include the extended multiplicity information of the ports, as it seems to improve the mappings. The alternative situation in this experiment, B, will be used as the new base situation in future experiments.

Additional Comments

Figure 7.3 illustrates, by comparing situations A and B, the effect of adding multiplicity information on how many base models must be mapped to find the best analogy in each of the 24 cases. The x axis represents base models in the list after retrieval, and the y axis the relative number of the number one analogies found from the 24 test cases.

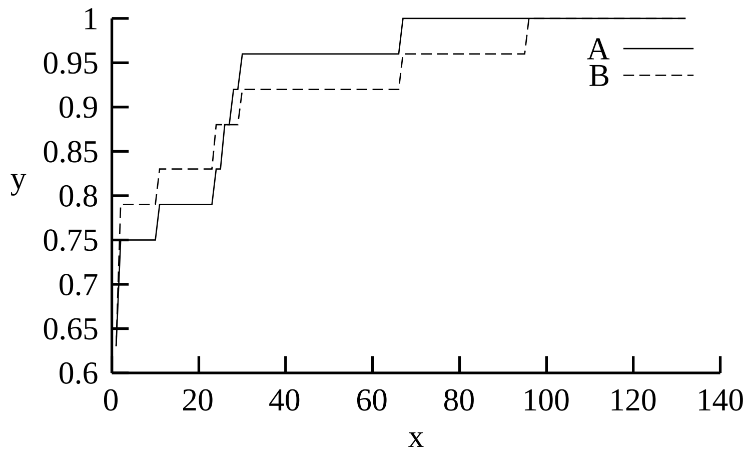


Figure 7.3: Fraction of best analogies found at position x after retrieval

In the first part of the list, B shows an improved situation over A. A possible explanation of this fact may be that in situation A the genetic algorithm fixes the structure too early due to greater semantic similarity. At about position 30, the number of analogies found in situation B is lower than in situation A. This poorer performance of B at higher positions may be explained by the fact that some mappings in A between ONE and MANY ports that were considered good, are now rejected, because of a lower score than a mapping between two equal types of ports.

An additional advantage stemming from the use of multiplicity information is seen when comparing two identical tests under situation B: All mappings were equal. We saw in section 7.2.3 that a little under 2 % of the mappings were different in two identical runs. This indicates that B gives a more stable behaviour of the genetic algorithm.

7.2.6 Adjust for Error in Sense Selection During Retrieval

Sometimes, when a software developer creates a new role model, it may be difficult to choose between meanings of a term. The problem is how to handle a situation such as the following: A software developer unintentionally selects the "wrong" sense for a specific role name. If this sense belongs to a WordNet hierarchy different from what the senses of the other roles it is compared to does, the semantic similarity value, sim_{sem}, between these roles equals 0.

To prevent a model from getting a too low score for a mapping when the name is the same, a small similarity value, e.g., $\gamma = 0.2$, can be assigned in these cases. Let wf_t and wf_b be the word forms that name the target and base roles, respectively, and sim_{sem} in these cases are calculated according to equation 5.10.

Notice that it is a question of only a small adjustment, performed only in situations when the mentioned condition occurs. Often this adjustment will have no influence on the upper bound of the semantic similarity between roles in the target and base models. One might think that this situation is not likely to occur, since another better match is found. This is not the case, however, as will be described below.

How Often Does a Role Get no Semantic Match During Retrieval?

When talking about the probability that two roles are semantically similar during retrieval, what it really means is the upper bound of the probability of a semantic match for the particular target role. It gives the similarity value *if* these two models were structurally mapped. Also, when searching for semantic similarity for the role names in the target model, the best map for several of the roles may be the same role in the base model. The value identified during mapping can in fact be a lot lower. This may indicate that the actual similarity model used is less important during retrieval than during mapping.

I selected one of the test cases with `Program committee job 2` as the target model. This model has 4 roles. When looking at the best matches between one of these 4 target roles and any role in the base models, there are the following occurrences:

- 28 identical matches

- 197 no match
- 307 partial match

In this particular example, no match was found in about 37 % of the situations.

It is reasonable to believe that when roles named by the same word form, but with meanings taken from different hierarchies, are compared, there may be no other roles in the role model that gives a (partial) match. To introduce an adjustment in this situation may therefore influence the outcome of retrieval. In situations where other role names have a greater similarity, this will have no influence on the result.

Adjustment Experiment

The base case is the alternative case from the test of addition of multiplicity information, i.e., situation B. In the previous experiment nothing involving the retrieval phase was changed. Here, I want to study the effect of modifications of the semantic similarity value on the ranking of role models after retrieval relative to the rank of the models according to analogy values. In a couple of models such differences in sense selection was inserted deliberately. For the rest of the models, I tried to avoid it.

I first analyzed the results of the mappings. To be able to measure such a small effect, the relevant cases are separated out. Matches between roles with the same name, but where the word meaning belonged to different term hierarchies, occur in 13 of the 24 test cases. These 13 cases were studied separately. There are a total of 34 occurrences of 5 different word forms. That this happens so frequently, even though I, for the most part, have tried to prevent it, indicates that this type of differences in sense selection is likely to happen.

For one of the target models, `Library1`, there occurs 5 times during mapping that a role is mapped to a role in a base model with the same name—same word form—but with different meaning, and where these meanings belong to a different word hierarchy. In 4 of these situations the best similarity value is 0. This implies that by correcting for a possible error in sense selection, the similarity would have been increased by 0.2 in these 4 cases.

I checked for the implications for the best analogies' positions after retrieval. For the 3 best analogies for each of the 24 cases, the only modifications were that the second and third best model had switched places a couple of times.

Conclusion It is not verified that this heuristic helps. No statistical significance is found. In some situations, where there is a deliberate choice behind the selection of name, such a modification may even make harm. In the following experiments, I decide *not* to adjust for error in sense selection.

7.2.7 Increase Weight on Structural Similarity During Retrieval

The purpose of this experiment is to see if good analogies will be ranked higher after retrieval when structural similarity is given relatively higher weight. This would be in line with theory saying that analogies are based on deeper, structural similarities rather than surface similarities, e.g., semantic information; see 3.1.2. If, in this experiment, base models with a high score during retrieval but with small analogy values are moved down in the list after retrieval, that experiment can be considered a success.

Modification is made by *reducing* the weight given to the semantic belief function. The experiment is conducted as several runs with the same set of target models, where, for each run, the weight of the semantic belief function is reduced before it is combined with the structural belief function. The weight given to the semantic similarity is 0.9, 0.8, 0.7, 0.5, and 0.0 of its original weight. The value 0.0 means that semantic similarity is given no weight.

Hypotheses The *null hypothesis* states that a reduction in weight on semantic similarity during retrieval will not improve the ranking after retrieval of the 10 best analogies.
The *alternative hypothesis* states that a reduction in weight of semantic similarity will improve the ranking after retrieval of the 10 best analogies.

The 10 best analogies for each of the 24 test cases in the two situations are compared as before. A change in x_{ij}^{y} is measured as $\delta_{ij} = x_{ij}^{B} - x_{ij}^{A}$. If the 10 best analogies are moved closer to the front of the list, the median of δ will be less than zero.

H_0**:** median $(\delta) = 0$
H_1**:** median $(\delta) < 0$

Analysis

The *sign test for median* is used to analyze the data, and table 7.5 shows the result. In this case there are more than one alternative situation B. B_k means that the semantic similarity in test B is reduced to k.

δ	N	Below	Equal	Above	P	Median
$B_{0.9}$-A	240	38	82	120	1.00	0.5
$B_{0.7}$-A	240	41	48	151	1.00	3.0
$B_{0.5}$-A	240	46	31	163	1.00	8.0
$B_{0.0}$-A	240	29	8	203	1.00	19.0

Table 7.5: Results from the experiment

The rank after retrieval of the best analogies is not improved by reducing the weight on semantic similarity during retrieval. On the contrary, it seems that the 10 best analogies have been given a lower rank after retrieval, contrary to the theory. The probability is 1 for all tests, and the null hypothesis can therefore *not* be rejected.

If I turn the problem around, use the same H_0 as before and change the alternative hypothesis to H_1: median $(\delta) > 0$, we get the following results if looking at only the two last tests in table 7.5.

δ	N	Below	Equal	Above	P	Median
$B_{0.5}$-A	240	46	31	163	0.00	8.0
$B_{0.0}$-A	240	29	8	203	0.00	19.0

Table 7.6: Testing for the alternative method giving higher values

It can be seen here that if the alternative hypothesis states that the increased weight structural similarity during retrieval would give a lower rank after retrieval to the best analogies, H_0 would have been rejected, i.e., H_1 : median < 0. The results even give some indication that the contrary could really be the case; that semantics is very important for identifying similarity among the models.

Figure 7.4 shows how many of the retrieved models must be passed on to the mapping phase to find the 10 best analogies for all 24 cases. The situation in the base case, A, with equal weight on structure and semantics is compared to situation $B_{0.5}$, where the weight on semantics has been reduced by half, and B_0 with no weight on semantics.

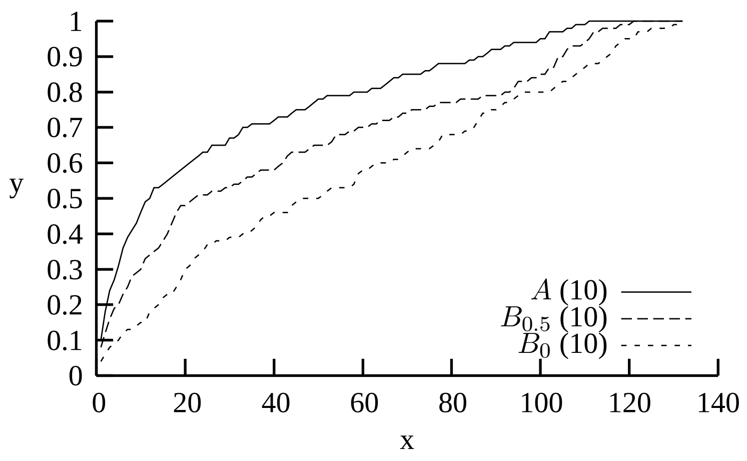


Figure 7.4: Fraction of 10 best analogies found relative to rank after retrieval

Since it may be unclear whether it is necessary to get the 10 best analogies, I made the same plot for the 5 best analogies for each case. See figure 7.5.

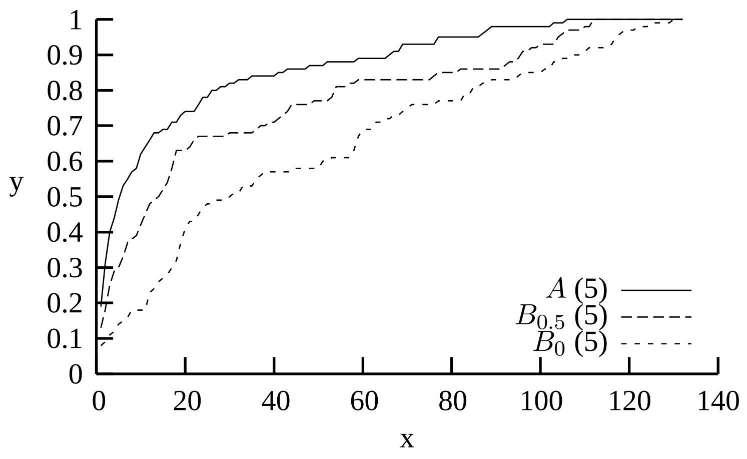


Figure 7.5: Fraction of 5 best analogies found relative to rank after retrieval

Figure 7.6 compares the fraction of the 5 best analogies found with that of the 10 best analogies for the three situations described above.

Conclusion H_0 cannot be rejected. A reduction in weight on semantic similarity during retrieval will not give the 10 best analogies a higher rank after retrieval.

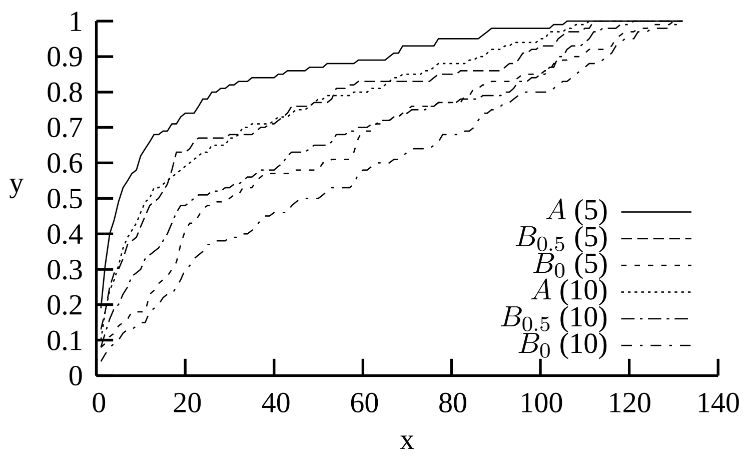


Figure 7.6: Comparison of situations with the 5 and 10 best analogies

7.2.8 Increase Weight on Semantics During Retrieval

The findings of the previous experiment indicates that semantics may play a more important role than structure during retrieval, at least for the current models in the repository. One reason to perform this experiment is that, with semantically similar models in the repository, it is more likely that they are reusable than are models that are only structurally similar. The argument is contrary to the arguments used in the last experiment. I set up an experiment where the weight on structure is reduced while semantics plays the same role as in the original case A.

Hypotheses The *null hypothesis* states that a reduced weight on structural similarity during retrieval will not improve the rank after retrieval of the 10 best analogies.
The *alternative hypothesis* states that by reducing weight on structural similarity during retrieval, the 10 best analogies will be given a higher rank after retrieval.

The 10 best analogies for each of the 24 test cases in the two situations are compared, giving the following results. The comparisons are done as in section 7.2.7. If the 10 best analogies are moved closer to the front of the list, the median of δ will be less than zero.

H_0**:** median $(\delta) = 0$
H_1**:** median $(\delta) < 0$

Analysis

The *sign test for median* is used on the 10 best analogies, and the results are given below. The base case A is the same as in the previous experiment, while the alternative case is called C. If situation C is better than situation A, the median will be smaller than 0.

δ	N	Below	Equal	Above	P	Median
$C_{0.9} - A$	240	119	89	32	0.000	0.0
$C_{0.8} - A$	240	128	74	38	0.000	-1.0
$C_{0.7} - A$	240	128	76	36	0.000	-1.0
$C_{0.5} - A$	240	138	65	37	0.000	-2.0
$C_{0.0} - A$	240	138	41	61	0.000	-3.0

Table 7.7: Effect of reducing weight of structural similarity

In table 7.7, δ represents situation C_k, where the value of k is the factor used to adjust the weight of the structural belief function. From the table it is seen that when reducing the weight on structure during retrieval, the best analogies get higher rank after retrieval. H_0 can thus be rejected.

Based on this result it is not possible to draw the conclusion that structure is not important. The reason for the result may be that the models used in the experiments are more similar semantically than structurally, or it may be that the structure description algorithm used is not able to capture the true structure of the models. The last point would probably be a greater problem for the larger models in the repository.

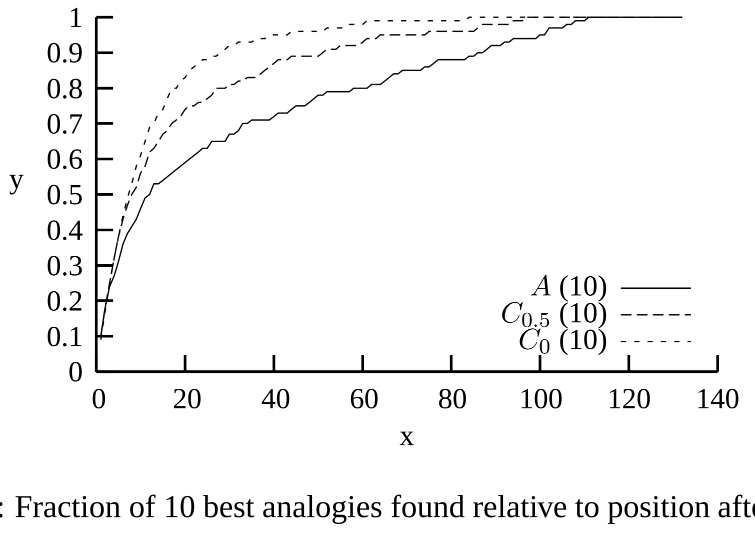


Figure 7.7: Fraction of 10 best analogies found relative to position after retrieval

Figure 7.7 compares the result when structural weight is reduced to 0.5 and 0.0 compared to case A. As can be seen, the best analogies are, to a larger extent, found among the first part of the list of models after retrieval. Figure 7.8 shows the similar situation for the 5 best analogies, and figure 7.9 compares the behaviour for the 10 and 5 best analogies.

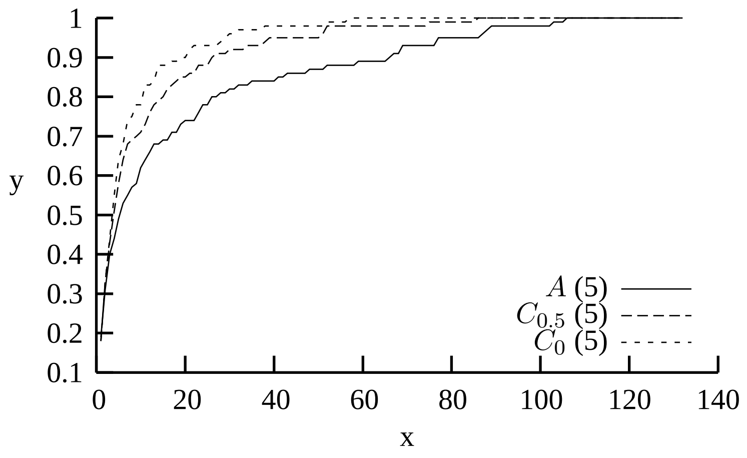


Figure 7.8: Effect of reducing weight on structure during retrieval of 5 best analogies

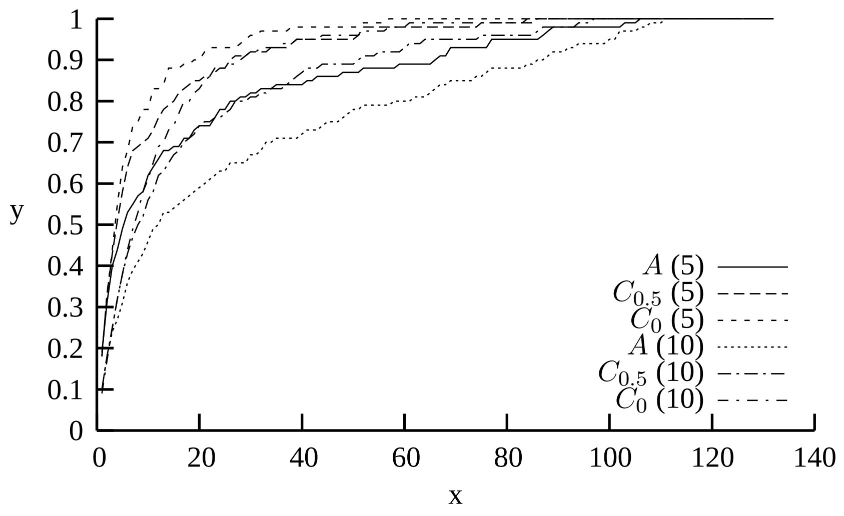


Figure 7.9: Comparing the results for the 5 and 10 best analogies

The analogy values are calculated using the same algorithms during all these tests, so this does not contradict the general theory that deeper, structural similarities are more important than surface similarities. The result may be that the algorithm used is not capable of capturing the structure of the models, or rather the difference among their structure, in a fruitful way.

As the two experiments have tried to show opposite effects, it is natural to show the effects combined in one figure. Figure 7.10 shows what fraction of the 5 best analogies will be found at a certain level of the list after retrieval for both the emphasis on structural and on semantic beliefs.

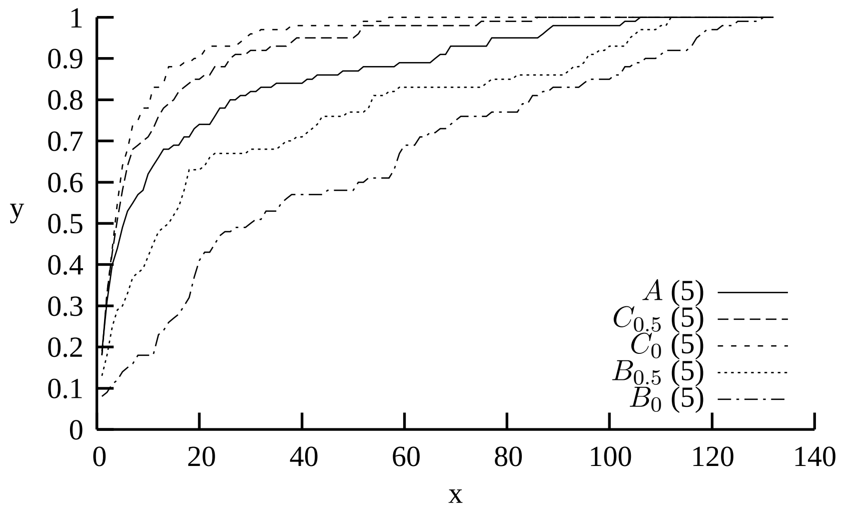


Figure 7.10: Comparison of effect of increase of structural and semantic weight for 5 best analogies

Conclusion This experiment showed that H_0 is rejected. Reduced weight on structural similarity during retrieval did give the 10 best analogies a higher rank after retrieval. The experiment indicates that the more weight on semantics the better. It also shows that the tendency to find the best analogies with a high rank after retrieval is strengthened if looking at the best 5 instead of the best 10 analogies.

7.2.9 Punish Large Base Models during Semantic Retrieval

One observation made is that some large base models that get a high score during retrieval, get a low mapping similarity. During semantic retrieval I identify the role from each base model giving the best semantic similarity to each role in the target model. If a base model has more roles than the target model, there is an increased probability that we can identify a good match, simply because there are a larger number of alternative role names to compare to. The model **program_committee_synthesis** is an example[1]. The complexity of the model, however, is more due to the number of ports than to the number of roles. It contains 16 roles and 36 ports.

[1]It is, as the name implies, a synthesized model, and it should probably not be included among the base models. However, in the experiments thus far, it has been included.

If a target model with only 4–5 roles is compared to this larger model, each of the roles in the target model has a higher probability to finding a role with a reasonable semantic similarity.

Some models may get an unreasonably high belief value during semantic retrieval, and thus the total retrieval, while the real analogy values are low. It follows that the goal is to the push weaker analogies down the list of base models m_1 during retrieval, while the top analogies are moved to the front of the list as much as possible. The base case, m_1, is the alternative case, B, from 7.2.5. Punishment is performed according to

$$\text{sim}_{sem} = \text{sim}_{sem} - k * \frac{i_b - i_t}{i_t}$$

where i_b and i_t are the number of roles in the base and target models, respectively. The experiment has been performed with $k = 0.05$ and $k = 0.1$. I call these experiments m_2 and m_3, respectively. The methods differ only in the way the sim$_{sem}$ is calculated. The base models are ranked according to analogy values, a_{ij}, as before, and their positions, x_{ij}, after retrieval are identified.

The two alternatives, m_2 and m_3, are compared to m_1. Each of the three situations is run for the 24 cases with different target model and with the same set of base models. I am interested in the effect this punishment has on the sequence of base models after retrieval.

Hypotheses The *null hypothesis* states that punishment of large base models with respect to their semantic similarity will not improve the rank of the 5 top analogies after retrieval.
The *alternative hypothesis* states that such punishment will have a positive effect on the rank of the 5 top analogies.

The effect is measured by taking the difference, δ, between a pairwise set of data from the base case, m_1, without punishment and the case, m_x, with punishment, where $x = [2, 3]$. The 5 best analogies for each of the 24 test cases for the two situations are compared, and the differences between the measurements are analyzed. One method is better than another if the best analogies are found higher up in the list after retrieval. If the alternative hypothesis is supported, the top analogies of m_2 or m_3 will be moved forward in the list, and $\delta > 0$.

H_0**:** median $(\delta) = 0$
H_1**:** median $(\delta) > 0$

Analysis

In some situations, punishment may affect the sequence of models in an unintentional way, as a large, true analogue will also be punished. If a synthesized model is found to be a good analogy, this model can be moved down the list, and the effect will be negative. The analogical reasoning engine, however, is supposed to be good also in situations where the analogies belong to a different application domain, and thus there is a tradeoff between these two goals.

For each of the x best analogies, its rank after retrieval is identified. The difference in rank after retrieval for these models is found. The intention is to reduce false similarity due to larger size of base models, with the risk of creating a reduced similarity in situations where a true analogy is being punished because the base model is larger than the target. It is unlikely that we get only examples of the first of these outcomes.

Sign test for median is used to analyze the data. I compared m_1 with m_2. For each position in the lists that are compared, a new value is calculated as $m_{1i} - m_{2i}$, where i represents the position in the list. If the value is negative, the model has been moved down the list in the m_2. I later did the same with m_1 and m_3.

When looking at the five best analogies for each target, and comparing m_1 with m_2 and m_1 with m_3, the results were

δ	N	Below	Equal	Above	P	Median
$m_1 - m_2$	120	28	76	16	0.9756	0.00
$m_1 - m_3$	120	29	71	20	0.9238	0.00

Table 7.8: Results from using the sign test for median on the punishment data

The probability of a result where m_2 or m_3 is better than m_1 in the two results listed above, is very small for both cases. When the two columns for m_1 and m_2 are plotted against each other, I get the results as seen to the left in figure 7.11, and to the right m_1 is plotted against m_3.

Figure 7.11 shows that of the 5 best models for all the 24 tests, only four get a considerably better position using method m_2, i.e., the one with a small punishment. These four are represented by the dots in the upper, left diagonal. Using m_3, only two of these models get a better position.

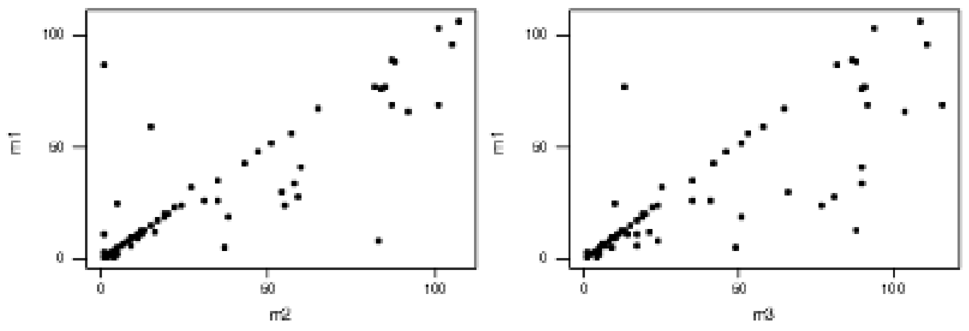


Figure 7.11: Method m_1 plotted against m_2 and m_3 respectively

Conclusion H_0 can not be rejected. There is no support in saying that punishment of large base models with respect to their semantic similarity will improve the rank of the 5 top analogies after retrieval.

Additional tests The same test was done with the 10 and 20 best base models. In these situations it is not so obvious that the models with punishment are worse than m_1. Since my goal was to try to find a model that was better than m_1, I can thus conclude that to punish large base models during semantic retrieval is not the way to go.

7.2.10 Remove Synthesized Models from Repository

The repository contains two synthesized role models. One of these was discussed in the previous section. These two models show up among the 10 best analogies in 10 of the 24 cases, although they are not given enough belief to be get a high rank after retrieval, and it would require a large number of role models to be transferred to the mapping phase to ensure that these are found.

When the project was first initiated, it was not intended that these models should be kept in the same part of the repository as the sub-models. The repository is designed with links from each role model to potential sub-models and synthesized models. Search should be done among the sub-models. If an analogy is found in the repository, the elaboration phase can include an analysis of the other models in the same project as the analogous model to see whether more of that project is reusable. I thought it was interesting, however, to see how the system would behave with such models present, as this shows how the algorithm degenerates when the sizes and structures varies more among the models.

One major problem concerning the synthesized models is that the algorithm identifying structural similarity cannot describe enough of their structure since I only use the three best structure descriptions, see section 5.2.1. The most important structure descriptions for a large model need not be the ones that will give the best match to a semantically similar model. The model **program_committee_synthesis** contains 36 ports, and the number of sub-structures in this model is far above the three that are stored in the repository. The sub-models from which these models are synthesized will also be kept in the repository, and the software developer can indirectly get to the synthesized models through them.

It is interesting to see what happens if these models are removed from the repository. This will give us a collection of models that is closer to the intended one, and the results stemming from structural similarity during retrieval may be fairer, and the best analogies may get a higher score after retrieval. Because there are only two synthesized models, I need not analyze the whole set of 24 cases, but only those in which one or both of these models occur among the best 10 analogies.

Hypotheses The *null hypothesis* states that, of the tests where one of the synthesized models were among the 10 best analogies, the rank after retrieval of the 10 best analogies are not improved by removing the synthesized models.
The *alternative hypothesis* states that the rank after retrieval of the 10 best analogies is improved after these models are removed.

In the base case, A, the synthesized models are in the repository, while in the alternative case, B, these models have been removed. The measurements of rank of the 10 best analogies for the two situations are made, as before, and $\delta_{ij} = x_{ij}^A - x_{ij}^B$ are calculated. The alternative hypothesis is strengthened if $\delta > 0$.

H_0**:** median $(\delta) = 0$
H_1**:** median $(\delta) > 0$

Results When the data from the 10 cases, where one or both of the synthesized models were included among the 10 best analogies, were analyzed using the sign test, the following result was obtained:

	N	Below	Equal	Above	P	Median
$A - B$	100	34	18	48	0.0756	0.00

Table 7.9: Results when synthesized models are removed

Conclusion H_0 cannot be rejected.

The submodels will still be removed from the repository in the remaining experiments, because the content that they represent—in the form of their sub-models—are present in the repository as such. The sub-models will also typically be more in line with the target model in size and complexity.

Additional tests If, instead, I look at only the 5 best analogies for each case, there are only 8 cases where A contains one of the synthesized models. When the other cases are removed, and I analyze the remaining data, we get the results as shown in table 7.10. The reduced set of A is named $A1$ and B is named $B1$.

	N	Below	Equal	Above	P	Median
$A1 - B1$	40	13	8	19	0.1885	0.00

Table 7.10: Result when the 5 best analogies are considered

In four of these cases one of the synthesized models ends up as the best analogy in situation A; see table 7.11. Here `rank` represents the rank after retrieval of the best analogy. The worst case, i.e., the lowest rank after retrieval of the best analogy is reduced from 96 to 65 when looking at these four cases when the synthesized models are removed, but the best analogies do not necessarily get a higher rank after retrieval, as can be seen in the table. No conclusion can be drawn based on this, however.

	A		B	
Target	rank	value	rank	value
CustomerPlaceOrder	96	0.585	26	0.508
Register for course	30	0.544	65	0.544
Program committee job no 2	24	0.803	1	0.781
Checkbook application	5	0.477	10	0.477
Median	27		18	

Table 7.11: The 4 cases where one of the synthesized models is the best analogy

Discussion One point to make at this stage is that the analogy values are low in these examples, except for the situation where a synthesized model is mapped to one of the models from which it is built. This is the case in the third line i table 7.11. The model that replaces the synthesized model here when it is removed, is a model called **IFIP example - no 1**, that is one example taken from the same application area. A last indication strengthening the belief in the results given above can be seen in figure 7.12 showing the fraction of the 10 best analogies from each of the depending on the number of models that are passed on from the retrieval to the mapping phase.

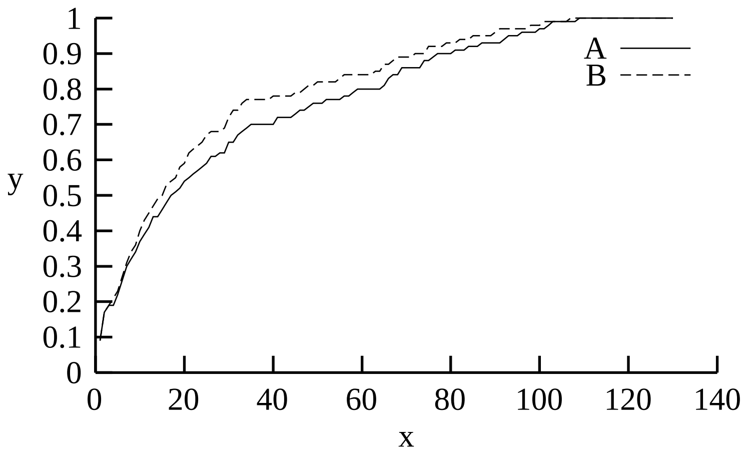


Figure 7.12: Fraction of 10 best analogous models found with and without the synthesized models

7.2.11 Use of GA Fitness Function to Rank Analogies during Mapping

The results found in sections 7.2.7 and 7.2.8, indicating that structure is of less importance than semantics during retrieval, is problematic since it contradicts current theory regarding analogical reasoning. One explanation could be that the analogy values are calculated with too much emphasis on semantics. They are calculated as average semantic similarity for the roles in the target model that are structurally mapped to a role in the base. Unmapped roles get semantic similarity equal to zero.

An alternative approach is to use the fitness function that is used by the genetic algorithm during mapping to calculate the mapping similarity value. This algorithm takes into account information on both structure and semantics.

The Experiment

The base case, A, for the experiment is the same as the alternative situation after the experiment in section 7.2.10. In A, the value of the analogy is calculated as the mean semantic similarity value of the roles in the target model that are mapped. In the alternative case, B, the analogy gets the value of the best genome returned from the fitness function. Because these two algorithms give very different values, it is not possible directly to compare the mapping similarity values. Instead, I want to see if B ranks the analogies in an order that is more in line with the rank of role models after retrieval.

Hypotheses The *null hypothesis* states that there will be no improvement in rank after retrieval for the 10 best analogies when the fitness function is used to calculate the mapping similarity.
The *alternative hypothesis* states that the use of the fitness function to calculate the mapping similarity will result in an improved rank of the 10 best analogies after retrieval.

For each of the 10 best analogies, a_{ij}, where i is its rank in mapping similarity for the test j, I measure the position of a_{ij} after retrieval. The difference, $\delta_{ij} = x^A_{ij} - x^B_{ij}$, is calculated for each of the 10 best analogies. Alternative B is better if $\delta > 0$.

H_0**:** Median $(\delta) = 0$
H_1**:** Median $(\delta) > 0$

Results When testing H_0 relative to H_1, I used a *sign test for median*. The results of this test for the 10 best analogies is shown in table 7.12.

	N	Below	Equal	Above	P	Median
B-A	240	162	12	66	1.0000	-18.5

Table 7.12: Results from using the fitness function to calculate the mapping similarity

Conclusion H_0 cannot be rejected. A majority of the 10 best analogies where given a lower rank than to a higher rank.

Further tests I ran the same experiment with varying weight on semantic and structural similarity like in the experiments in sections 7.2.7 and 7.2.8. When reducing the weight of semantic similarity belief, the results were worse, just as was the case in section 7.2.7. When reducing the weight of structural similarity belief, the situation improved. These results are in line with the experiment in section 7.2.8.

If comparing situation B, where B is the alternative situation described above in this section, with the situations $B_{str_{0.5}}$ and B_{str_0}, meaning that the weight on structure has been reduced to 0.5 and 0 respectively, and where I have looked at the 10 best analogies for each case, I get the results as shown in table 7.13.

	N	Below	Equal	Above	P	Median
$A - B$	240	125	27	88	0.9954	-2.0
$A - B_{sem_{0.5}}$	240	121	26	93	0.9763	-1.0
$A - B_{sem_0}$	240	102	26	112	0.2692	0.0
$A - B_{str_{0.5}}$	240	141	26	73	1.0000	-3.0
$A - B_{str_0}$	240	159	25	56	1.0000	-6.0

Table 7.13: The effect of reducing weight on structural similarity belief

7.3 Discussion

Some analogies was intentionally added to the repository. Some wholesaler and library models were created for purpose. Some of these models have been shown before. Figure 7.13 shows an example with mappings between the roles. The structure and size of the two models is equal and two of the roles have the same name. This wholesaler model was ranked first after retrieval due to the high structural and partly semantic similarity. After mapping this model is ranked as number three, with a mapping similarity of 0.7987. This example illustrates that it is possible to identify analogies across domains.

One question I wanted to answer was how many base models should be passed on to the mapping phase to have a fair chance of identifying the best analogies in all cases. In the first experiment (see section 7.2.1) the aim was to establish a base situation. There, if 26 of 132 base models are passed on to the mapping phase in each case, the best analogy is found in 21 of 24 of the cases. All but one of them will be found if 30 base models are passed on

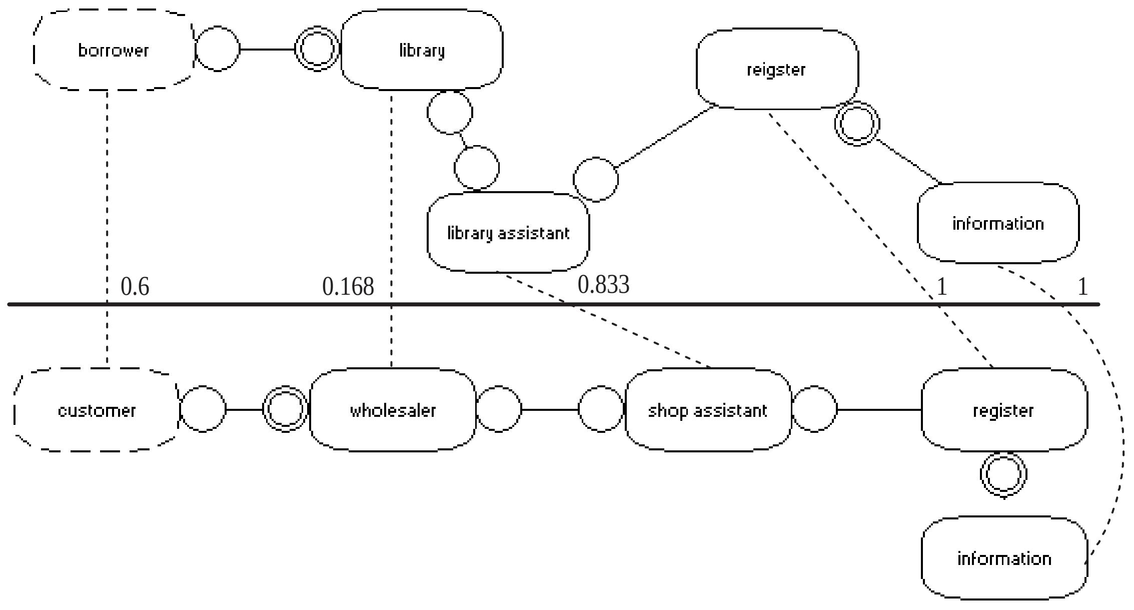


Figure 7.13: Mapping of models from different application domain

to mapping. For the last case, the best analogy gets a low score because of a low structural mapping; see discussion in section 7.2.1. Figure 7.14 shows the mapping. Also note that the mapping between the two roles *library assistant* in the models does not give a value of 1. This is due to the fact that in one case a specialization of *employee* is used, while in the other a modifier of *assistant* is used; see also figure 7.2. The semantic similarity is therefore lower.

The base model, the lower model in figure 7.2, is ranked as number 123 for structural similarity, while it is ranked number 6 in terms of semantic similarity. Both models have four roles. Even though they have different structure—a tree and a sequence—all the roles are mapped due to high semantic similarity. This contributes to the result that makes this base model the best analogy. If *library assistant* had been coded equally in the two cases, the probability of semantic similarity would have been 1, and the model would have been ranked higher after retrieval, but the structural rank would be the same.

In the experiment where the fitness function is used to calculate the mapping similarity (section 7.2.11), this particular base model is ranked as the 8^{th} best analogy. This is because the semantics alone is not used to calculate the mapping similarity. The experiment shows that this approach does not give better overall results. Another alternative to reflect that the analogy between these two models is week, could be to adjust the mapping similarity based on

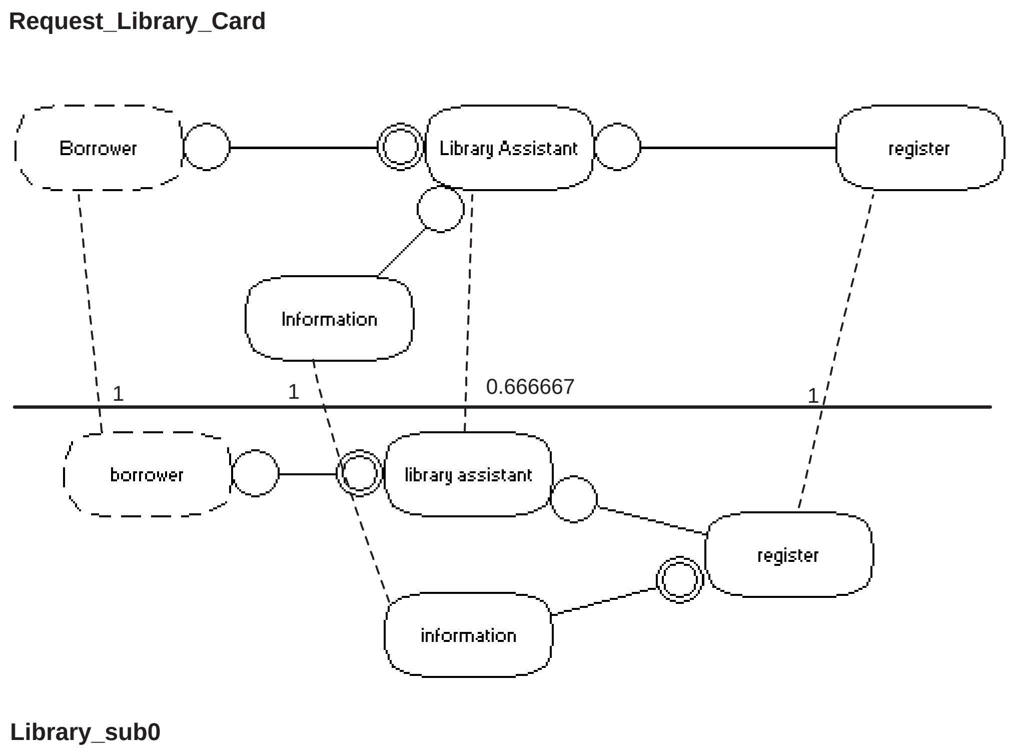


Figure 7.14: Mapping of models with unequal structure

the fact that the structural similarity is not so close. It is not advisable, however, to make the algorithms too complicated as I do not know much about the intended purpose of this particular role model. Such information should be found in a textual description that the user can view after the model has been selected, or it could be found by analyzing other properties of the same model. An argument for allowing mapping of roles that are not mapped structurally like in the above example, is that the software developer may have made a mistake when making the new model, and that this model therefore is suggested as a possible match. All in all, the base case can be called a promising result.

A concluding summery of the remaining experiments is:

- When mapping a target model onto itself the analogical engine is in fact capable of identifying itself as the best analogy. This experiment was included to test the soundness of the engine.

- The algorithms were also tested for stability by performing two identical runs. Of the 3165 mappings made in this case, only 5 differed in value. This amounts to less than 0.2 %, so in general I can eliminate this error.

- Introduction of a modifier does not change the values of mappings, so the only effect that is seen on the mappings is if two roles being compared has the same name but these names are coded differently—one with a modifier and one by adding a new *word form* to the repository.

- The introduction of multiplicity information to the role models gave the result that the 10 best analogies were ranked higher after retrieval. This also resulted in more stable mappings.

- The experiments with varying weights on structure and semantics during retrieval showed that for the models in the repository, semantics seems to be more important than structure. This may only imply that for the content of this specific repository, the are models that more semantically than structurally similar. The important conclusion I can draw, however, is that the user should be given the freedom to balance the emphasis on structure and semantics during retrieval. The default should be a neutral balance. The user may know that the repository contains models from the same application area and cause him to emphasize semantics.

- The fitness function is used to rank the analogies to test whether the results from the previous experiments on weighting semantics and structure during retrieval was the result on bogus analogy values. However, the results show that no matter how structure and semantics were combined, the ranks of the analogies were not more in line with their rank after retrieval.

To give a stronger conclusion, one would have to perform experiments where users actually evaluate the mappings that are suggested, or alternatively try to reuse an earlier design based on the analogous analysis models. Such experiments have not been performed as part of this this thesis.

Based on the experiments performed, I was not capable of setting a definite lower threshold for the mapping similarity, although the default value of 0.6 should, in most cases, get potential analogies included. To more precisely find such a lower threshold, an experiment could be set up where users get a set of models with known similarity values based on experiments, and where they are asked to identify which models are analogous. I will, however, not involve users in any experiments, so the measures used here for a good analogy is one where the calculated mapping similarity is as high as possible. A user may then accept or reject a proposed analogy.

7.4 Validity Evaluation

When conducting experiments, their validity should be considered. We normally distinguish between internal and external validity, and sometimes also conclusion, and construct validity (Wohlin, Runeson, Höst, Ohlsson, Regnell, and Wesslén, 2000).

In theory testing, the priorities of validity testing for experiments is in increasing order: internal, construct, conclusion, and external validity testing, while in applied research external validity is more important. My research falls somewhat in between, as it is applied in the sense that I test an approach on a set of models that are written with the intention of being real-world, but I cannot claim that I have tested whether my results are valid for a wider set of neither models, application areas, nor model types.

7.4.1 Conclusion Validity

Conclusion validity is concerned with the relationship between the treatment and the outcome. Can I be certain that there is a statistical relationship, i.e., with a given significance. Wohlin et al. (2000) divides this in statistical power and reliability of measures.

Statistical Power I use two different statistical tests, Sign test and Wilcoxon rank test. The choice of test is restricted by the fact that the data used do not show normal distribution, so a non-parametric test is required. The data, difference in analogy values and difference in rank after retrieval, can be found on an ordinal scale.

Reliability of Measures When I sometimes get varying results in different analogies even though no change in the algorithm have been made, this is an error source. I did two tests to check the variability when conducting two identical tests: for the original case and when multiplicity information was used (see section 7.2.3). In the first case the variability was of a magnitude less than 0.2 %, while in the second the results were identical.

In other experiments, however, where I wanted to improve the rank after retrieval of good analogies and had made a change in the algorithm accordingly, differences also occurred. I have not been able to explain these differences other than by the fact that the genetic algorithm creates genomes at random and therefore may reach an alternative local optimum. They do not occur frequently among the better analogies, however.

7.4.2 Internal Validity

Internal validity is concerned with whether an observed relationship between the treatment and the outcome is a causal relationship, and that it is not a result of a factor of which I have no control or have not measured. It is discussed in terms of instrumentation.

Instrumentation One important threat is the correctness of the program used to identify the analogies. The calculations of structural descriptions, have been checked manually for correctness. This also applies to the calculations of semantic similarity, both during retrieval and mapping. As far as I can observe, the calculations are correct. It is not possible, however, to remove completely errors in a program.

The role models that are represented in the repository constitute the objects of study. An important question is whether the role models that where chosen as *target* model in the study are representative for the models currently in the repository. The table below shows some general properties of the models' structure and size (in terms of number of roles). Recall that each model is described by, at most, three structure descriptions. Large models, with more complex structure, will therefor not be fully described during retrieval.

Table 7.14 shows the percentage of each of the types of structures as well as the percentage of lacking structures. The last number is to indicate how many sub-models that have less than three sub-structures.

The structures in table 7.14, chain, star, tree, and ring, are listed in increasing order according to complexity. The size of the target models is not less than the average of the models in the repository. The selected target models are simpler, however, than the average models in the repository, i.e., there are more occurrences of the simplest structure—the chains—than in the average repository model. There are also more target models with less than three structure

	Models		
	All	All except synth.	Target
Mean size	5.78	5.6	6.08
Median size	5	5	5
Chain (%)	44.4	45	52.8
Star (%)	17	16.8	12.5
Tree (%)	24.8	24.4	20.8
Ring (%)	5.3	5.1	2.8
Lacking structures (%)	8.5	8.7	11.1

Table 7.14: Aggregated properties of all models and target models

descriptions. This could indicate that the targets are not representative for the models in the repository. It is reasonable, however, to argue that when a systems developer searches for an analogy for a particular target model, then the model is unfinished and one of the goals is to identify an analogous models from which some properties also can be transferred.

7.4.3 Construct Validity

In my case, I look for better analogies, and I have defined this to mean a higher mapping similarity, although it is clear that I cannot claim that this implies that the analogies are actually useful due to lacking expert evaluations. Since I have not tried to reuse the analogies, and since there is no implementation of the systems being modelled, I have no proof that the suggested analogies are in fact reusable.

7.4.4 External Validity

External validity is concerned with generalization. If our results are externally valid, it should be possible to generalize the result of a study outside the scope of the study. The purpose of this research has not been to conclude on this issue.

The intention has been to find models that are as realistic as possible. Many of the role models in the repository has been collected from books and papers on software development. An additional few have been created by participants in the ROSA project with the intention of being analogous. Some of the models are variants of other models. No models are developed as part of a real software development project and later implemented. The question is, are the models in the repository representative of the models that could be generated by software developers. This we do not know.

The size and structure varies among the models. This would probably also be the case in a realistic repository. I have selected target models that represent the various types of described structures and varied the combination and size of them.

The genetic algorithm works on a graphical representation of a model, so any model that can be converted into a directed graph could in principle be analyzed using the techniques presented in this work. Examples of such models are for example UML 2 sequence diagrams. In order to give a more complete answer to this question, more research needs to be done.

Chapter 8

Summary and Conclusions

The main research question for this research project has been [1]:

Is analogical reasoning a viable approach to identifying potentially reusable artifacts from early phases of the software development life cycle?

Examples of such artifacts are OOram role models. The above question is so general that it is difficult to answer it precisely and is therefore operationalized as follows:

OPERATIONALIZED RESEARCH QUESTION: *Is analogical reasoning a viable approach to retrieving potentially reusable OOram role models from a test bed of models?*

Three sub-questions, that are easier to address, are formulated, and the answers to them will together shed some light on the main question. They are:

Q 1 What kind of information is relevant to identify analogies between role models?

Q 2 How can we use this information to realize both retrieval and mapping of analogies?

Q 3 How can we balance this information in our identification of analogies?

In section 8.1, the findings related to these sub-questions are discussed separately. In section 8.2, further work is discussed.

8.1 Summary

The discussion in this section refers partly back to the analysis regarding the use of WordNet in chapter 4, the similarity model in chapter 5, and chapter 7 where the experiments are discussed. The three sub-questions are here discussed separately.

[1]The results of this thesis is also presented in Bjørnestad (2003)

8.1.1 Q 1: What kind of information is relevant

If the software developer can identify reusable analysis models from a base of such models without investing a lot of time in describing his new model by use of AR, this would be an important time saving opportunity. When creating OOram role models, one starts creating the roles and the communication paths between them. Later, cardinality information as well as role attributes and methods are inserted. An important question is whether it is possible to identify analogies based on knowledge of role names and communication paths among these roles. Experiments were set up to find analogies among role models with only such information.

Chapter 4 discussed how the semantics in the ROSA repository should be organized based on the lexical database WordNet. It demonstrated that the structure and content of WordNet makes it suitable for the ROSA term space. It has been demonstrates, through examples, how the semantic similarity model can be used to identify similarity between terms where others have previously argued that there is none.

Section4.3.3 Various ways to handle situations where a role name does not exist in the term space have been discussed, and two of these alternatives are used in the experiments; 1) the use of modifiers in OOram role names, and 2) the possibility to let the user add new word meanings to the term space. I argue that the first alternative can give enough semantic strength, while at the same time ensuring a correct handling of semantic similarity. This point is discussed further in section 5.3.5, and it is shown through experiments that the use of modifiers handles this situation to satisfaction.

The main problem with using WordNet for naming OOram roles is that the systems developer must choose among the different word meanings of a term. This is time consuming and can be error prone, and the software developer should be given all possible help in this process. In WordNet itself, the most common meaning of a term is listed first. While this has not been enforced in the ROSA repository, this could certainly be done. The glosses, i.e., explanations of a particular word meaning, and often also examples of use could be used as an aid when selecting a word meaning. Another solution might be to look at what other word meanings have been used in existing models in the repository. Given the organization of the repository, such information is easy to get.

In OOram role models we distinguish between the number of occurrences of the roles at the other end of a path a role knows about, e.g., *none, one*, or *many* (this is similar to specifying cardinality in conceptual modelling). Such situations will be handled quite differently in the implementation. This can for example distinguish between a role that acts as a container and one that is merely storing information about one item, e.g., a document. This information is therefore important for identifying models that are structurally more equal.

The experiment testing the importance of multiplicity information on the ports shows that the mapping similarities are more stable than without distinguishing between them although the actual ranking of analogies is unchanged. This option is kept even though the results of the experiment was not statistically significant.

8.1.2 Q 2: How can we use this information for Retrieval and Mapping

The two semantic similarity models that were proposed in section 5.3 are both capable of identifying semantic similarity among roles based on the distance between the word meanings used. The difference between these models is related to whether they take into consideration the level in the WordNet hierarchy, and if the word meanings that are found at higher levels in the tree can be considered to be semantically more closely related. There are some indications of this, and the selection of a semantic model, especially for use during retrieval, may be based more on pragmatics, as the semantic belief computed represents an upper bound.

The chosen similarity model for the experiments measures the distance between two synonym sets in the term space, but it also takes into account the difference in abstraction level of the two synsets. Two terms at about the same abstraction level are more likely to be interchangeable than two terms with greater difference in abstraction level, even though the distance between the terms in the two pairs is the same. In the experiments performed thus far, the synsetID has been coded to distinguish among word meanings. When this approach is used by systems developers, the word meanings should be selectable through a user interface where the senses, explanations, and examples of use, if they exist, are displayed.

Analysis and experiments performed as part of this project show that it is possible to use analogical reasoning to identify OOram role models that are similar to a target model. The suggested approach supports both the retrieval and mapping phases of analogical reasoning.

The results of a mapping between two models from different application domains also show that the approach is capable of identifying such analogies although in general we can not conclude that the models that are found are really analogous.

During retrieval, structure descriptions are used to identify structural similarity. This does not give a complete picture of the structures of the models, but fulfills the intention to get a quick identification of those models that have similar structures. The semantics in ROSA is modelled after WordNet as this ensures that there is a large vocabulary modelled by expert lexicographers. Semantic retrieval identifies an upper bound for the semantic similarity. Structural and semantic similarity are combined in a belief of whether a base model is analogous to a target. The rank of base models after retrieval decides what models are passed to the mapping phase. During mapping, a genetic algorithm was used to identify analogies based on structural and semantic similarity. The models that were mapped should, after retrieval, be ranked as similarly as possible to the rank they get after mapping. Mapping similarity was calculated based on an average of the semantic similarity for the roles in the target model. The potentially best analogies were found among the 26 highest ranked base models after retrieval in 21 out of 24 cases.

Many base models may therefore safely be removed from the mapping. Still, it is unclear how this will evolve as the repository becomes larger. In the current repository it means that about 50 % of the models must be included to find the best analogies in all the 24 cases, but it is uncertain if this percentage will be the same in a larger repository. It is not possible, however, safely to cut the number of models sent to mapping to 20–30 and still be reasonably confident that the best analogy will be found in all situations.

8.1.3 Q 3: How can we balance information in identification of analogies

Theory within AR claims that the underlying structural similarity is more important than surface similarities for identifying similarities. This is the theory on which the algorithms were built. Some of the experiments indicate, however, that if semantic similarity is emphasized more than structure during retrieval, the best analogies will be ranked higher, meaning that fewer models need to be transferred to the mapping phase. These results indicate that the system should be somewhat flexible on this point, permitting the systems developer to move the emphasis in one direction or the other. He may have knowledge about some previous projects in the repository and take advantage of such knowledge.

The algorithm that calculates mapping similarities is based on the semantic similarity values of the structurally mapped roles of the target model. To test whether this leads to a too strong weight on semantics, an experiment was run where the GA fitness function was used instead. This algorithm did not result in an improved result when it comes to the rank of the best analogies after retrieval, and this algorithm was therefore abandoned.

In section 4.2.1, the semantic requirements of the ROSA analogy machine was described, and section 4.2.2, discusses the semantics of OOram role models. Based on the collaboration view of OOram models (see appendix A), information of the role names was the most important. The experiments seem to identify the best analogies in most cases.

8.1.4 Conclusion

It has been shown through a set of experiments that the proposed approach can identify, during analogical retrieval, the OOram models with highest mapping similarity, in most cases. In all cases the best analogies are found if only 50 % of the models are sent to the analogical mapping phase.

Based on theory from analogical reasoning, the algorithm was based on an equal weight on semantic and structural information. The experiments showed that better results were achieved when more emphasis was put on semantics. This can be true when there exist models from within the same application domain in the repository. Many software companies have special niches, so it is reasonable to think that this will be the case in such organizations. The developer should therefore be given the opportunity to determine the balance between the two. Although the model repository is not as large as can be expected from a working repository in a software organization of moderate size, it is large enough to draw conclusions about the relevance of the approach.

All in all, the suggested approach to integrate AR in a software development tool seems to be viable. It has been

- identified information that can be used to search for analogies
- demonstrated how both the retrieval and mapping phases of AR can use this information

- demonstrated that the suggested approach is capable of identifying the best analogies, i.e., the models with highest mapping similarity, among the highest ranked models after retrieval.
- demonstrated how the balance between the semantic and structural information can be done, and discussed in what situation one or the other of these types of information might turn out more favorable than the other.

These items together answers the operationalized research question in a satisfactory way.

This work is an attempt to combine retrieval and mapping and such efforts are not common. Section 3.1 presents attempts to use analogical reasoning to identify reusable software components. The examples described here did, however, not combine retrieval and mapping. CBR, as discussed in section 3.4, has focused more on retrieval, and there is a lot to be learned from this.

More work is required to adjust the algorithms and develop environments that give the user more control, although sensible default values for the parameters for the analogy engine should be offered. Also, it might be very interesting to see whether the use of information about the behavioural aspects of the systems might turn out more useful for identifying analogous role models than just structural information as has been demonstrated here.

But the question remains, does this imply that a conclusion can be drawn concerning the main research question? Only to the extent that we accept that a high mapping similarity is the same as good analogy. Further study is needed to determine this.

It has not been shown that models identified as analogous to the target models are in fact reusable. However, the main research question only states that they should be potentially reusable. It is a good question, though. To determine whether an analogy (or a model with good mapping similarity) is reusable, one would have to implement the system to represented by the base model, and try to use the implementation in building the target systems. The question, however, concerning whether the models are really reusable, is left to further research.

8.2 Future Work

There are three main fields within which further work should be directed:

1. evaluate the results of this work, where software developers are asked which of the models are identified as analogies that they actually believe are analogous.
2. study whether the artifacts from later stages of the development process are
3. test and extend algorithms to improve the AR technique, reusable

These directions will be discussed in the next sections.

8.2.1 Evaluation of the Analogies

Thus far the approach has not been evaluated from a user's perspective. I have taken the perspective that a model that is structurally and semantically similar is analogous to the target model. How a systems developer will view this should be analyzed. An experiment could be set up where a set of software developers was given the task of sorting through all the base models—on paper—to find the ones they consider to be most analogous to a given target model. If they were given the same set of target models that has been used in the experiments in this thesis, and it they where to pick the 5 best candidate for each target model, the comparison would be straightforward.

Such evaluation might also help to solve the problem of what mapping similarity value will be the lower bound of what is interesting to show to a user. It may also answer the question of what should be set as a lower bound on the retrieval similarity for a base model after the retrieval phase.

8.2.2 Transfer of Information from the Analogy

Even though an analysis model is considered analogous, it is not necessarily the case that artifacts from design and implementation are reusable. To study this problem, a known case with a known design and implementation should be available in the repository. A new case, where one or more analysis models are considered analogous, should be created. Artifacts from the analogous base case should be attempted transferred to the new case. Obviously not all artifacts can be transferred, but it would be interesting to see how much, if anything, of the original case can be reused.

8.2.3 Improvements and Extensions

Test Structure Descriptions

The results indicating that structure should be given less emphasis than semantics during retrieval, calls for further study. One direction this could take is to look at the structure descriptions themselves. For larger role models a problem can be that not all roles in the models are covered by the structure descriptions for that model.

One possibility could be to make it possible to allow more than three structure descriptions for each model. This will reduce the efficiency of the retrieval, but if the result is better, it may be worth it.

Reuse of Scenario Views in OOram

In the implementation and experiments presented in this thesis, the OOram role model collaboration view is used. This view emphasizes the roles and connections among them. An alternative would be to use the scenario view that emphasizes the message flow between the roles; see section A. One could argue that such a view gives more information about the intention of the model at hand. Alternatively, these views can be presented to the user after an analogy has been identified.

Use Suggested Approach with Other Modelling Techniques

Today, UML is the most wide spread object-oriented modelling technique. For the ROSA approach to reuse to be accepted in the industry, I would have to show that the technique is applicable with UML. The type of UML model that is most comparable to the OOram analysis models, is the collaboration diagram.

The *actor* in the UML diagram would play the role of the *stimulus role* in OOram. It is clearly stated what direction the messages flow, but while the collaboration diagrams can also show a numbered sequence of messages, the OOram models do not. The use of *multiobjects* in UML is parallel to the use of MANY ports in OOram.

UML collaboration models can be converted to a directed graph with the actor being the root node. As long as this requirement is maintained, the implementation should be easily modified to be used with this type of models. An extension in this direction could be done as a refactoring to allow for several types of modeling techniques being supported. It could be done by introducing a generic type of modelling technique as an abstract factory (Gamma et al., 1995), and by having classes related to the special modelling techniques as subclasses.

Sequence Models or Scenarios

In all the experiments performed thus far, only the role names have been used. Another direction in which to move could be to use the OOram Scenarios or UML sequence models when trying to identify analogies. In this case both the role and message names would be used. Graphs could be created where a stimulus role or an actor initiates the activity, and both role and message names would be represented as nodes in the graphs. Every other node would be a message and the others would represent roles. This is possible to do. The question is whether the dynamic aspect of the models is more important for reuse than are the static properties of the role models.

A more flexible way to do both retrieval and mapping is to see what information has been inserted into the target model and what views have been used, and then set up the analogy search in relation to this. If the systems developer has specified structural aspects of the role model, this is the basis for search. If, on the other hand, behavioral information is added, this part is also included in the search.

Although no experiments have been performed in this direction, it certainly would be an interesting approach. It is doable, as the type of nodes—role or message names—could be distinguished among, and as the structure of the verb and noun part of WordNet is the same, it would not require a restructuring of the ROSA term base. As discussed in section B.3, method names could have an extension in addition to a verb, i.e., *add name*, the message modifier would be placed after the main name whereas it is placed before the main name for the role names. Plain polymorphism could be used to solve this problem in the implementation, as the name of the OOram components is placed in a common superclass of the **OOramRole** and **OOramMessage** classes.

Problems Related to Choosing a Particular Word Meaning

The use of a lexical database such as WordNet as the basis for the ROSA term space is advantageous. The system developer avoids the problems of making decisions about how the terms are related. The system developer must select among different meanings of a particular term. This may be a problem. In cases where the name of an OOram component is not already in the term space, the name must be split into terms. For this part of the system to be useful for a systems developer, more work needs to be done to automate this process. This is definitely viable.

The current version of the ROSA term space is dependent on WordNet synset identifiers that are generated as the WordNet database is created as an offset into the WordNet term file, and they will therefore probably vary among versions of WordNet.

Another limitation has been the availability of analysis models. Several models included in the repository are added only as *noise*, and they are definitely not all at the same abstraction level.

Domain Analysis

Although I have not considered using domain analysis, it is clear that the chosen approach would benefit from its use. WordNet was developed to model the vocabulary of a person with English as the first language. This is hardly what is needed in a CASE tool, where there is a need to have a vocabulary of a more technical character directed towards the application domains with which the system should be used. This should be something that can be modified by the users of the system. If an ontology exists for a particular domain, and it could be provided in terms of the existing WordNet data base, that type of modifications will be easy. One way of using the analogy mapping in ROSA could be to create a generic model based on a very good analogy between two or more role models. This is similar to what is described as an alternative of creating a domain model (Neighbors, 1989).

Tool Support for Sense Selection

The tool used by a systems developer should support the task of selecting meanings of a particular term. When the user is adding a name for a role, the alternative synonym sets with definitions and examples, if provided, should be displayed. This will make it easier to see which meaning to choose. Information about what other models have used the particular word meaning could also be displayed at this point, as there is a direct link to the models that use the different word meanings.

One of the problems related to use of a semantic database with polysemy is that the user must choose among the different meanings. WordNet contains word forms with at least 29 meanings. One means of reducing this problem is to create the list of meanings for a word form as a self-organizing list. This means that the word meanings that are used most frequently are placed first in the list. The original organization could be based on the frequency information that for many words already exist in WordNet. As the repository is filled with projects, the meanings that are used most frequently, will show up first in the list a user must select from.

The background for this suggestion is a belief that many of the meanings in WordNet will not be relevant for an application such as the ROSA Tool. As I currently have no definite knowledge about this, however, it would be too drastic to remove them. This again implies that at a later point of time, when there is more experience with the tools, tests examining what word meanings are actually used, may also be performed. It could also be interesting to study what synset trees are are used most frequently in software development. At that point in time, it may be reasonable to remove parts of the term space that is not used.

Bibliography

H. Adachi, Y. Kobayashi, and T. Ohta. Software design support using case-based reasoning. In *Proc. of International Conference on Expert Systems for Development*, pages 85–90, 1994.

C. Alexander, S. Ishakawa, M. Silverstein, M. Jacobsen, I. Fiksdahl-King, and S. Angel. *A Pattern Language*. Oxford University Press, New York, 1977.

Scott W. Ambler. *The Object Primer: Agile Model Driven Development with UML 2*. Cambridge University Press,, 3 edition, 2004.

Egil P. Andersen and Trygve Reenskaug. System design by composing structures of interacting objects. In *ECOOP'92, Proceedings of the 1992 European Conference on Object-Oriented Programming*, LNCS 615, pages 133–152, July 1992.

Guillermo Arango. Domain analysis methods. Ellis Horwood Workshops, chapter 2, pages 17–49. Ellis Horwood, 1994.

Kevin D. Ashley. Arguing by analogy in law: A case-based model. In D.H. Helman, editor, *Analogical Reasoning*, pages 205–224. Kluwer Academic Publisher, 1988.

Ricardo Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley, 1999.

E. Ray Bareiss, Bruce W. Porter, and Craig C. Wier. PROTOS: An exemplar-based learning apprentice. *International Journal of Man-Machine Studies*, 29(5):549–561, 1988.

Victor R. Basili. Software development: A paradigm for the future. In *Proc. of the 13th Annual International Computer Software and Applications Conference, COMPSAC 89*, pages 471–485, 1989.

Roberto Bellinzona, Maria Grazia Fugini, and Barbara Pernici. Reusing specifications in oo applications. *IEEE Software*, 12(2):65–75, March 1994.

S. Berczuk. Finding solutions through pattern languages. *Computer*, 27(12):75–76, December 1994.

Lucy M. Berlin. When objects collide: Experiences with reusing multiple class hierarchies. In Norman Meyrowitz, editor, *Proceedings of the Conference on Object-oriented Programming Systems, Languages and ApplicationsEuropean Conference on Object-Oriented Programming*, pages 181–193, Ottawa, Canada, October 1990. Association for Computing Machinery, ACM Press. Published as ACM SIGPLAN Notices Vol. 25, No. 10, Oct. 1990.

Ted Biggerstaff and Charles Richter. Reusability framework, assessment, and directions. In Biggerstaff and Perlis (1989), chapter 1, pages 1–17.

Ted J. Biggerstaff and Alan J. Perlis, editors. *Software Reusability. Vol. I. Concepts and Models*. Addison Wesley, 1989.

Andreas Birrer and Thomas Eggenschwiler. Frameworks in the financial engineering domain: An experience report. In Oscar M. Nierstrasz, editor, *ECOOP'93*, LNCS 707, pages 21–35, Kaiserslautern, Germany, July 1993. Springer-Verlag.

S. Bjørnestad, B. Tessem, K.M. Tornes, and G. Steine-Eriksen. Useful characteristics for identifying analogous specifications. Technical Report No. 24, ISSN 0803–6489, Dept. of Information Science, Univ. of Bergen, 1994.

Solveig Bjørnestad. Software reuse, what, when, and how? In *IRIS'24, Proceedings of the 24th Information Systems Research Seminar in Scandinavia*, volume III, pages 130–144, Ulvik, Norway, 11–14 Aug 2001. Dept. of Information Science, University of Bergen.

Solveig Bjørnestad. Analogical reasoning for reuse of object-oriented specifications. In K. D. Ashley and D. Bridge, editors, *Case-Based Reasoning Research and Development, Proc. of the 5th International Conference on Case-Based Reasoning, ICCBR 2003*, volume 2689 of *LNAI*, pages 50–64, Trondheim, Norway, June 23-26 2003. Springer Verlag.

Solveig Bjørnestad. Relations to support reuse of specifications. In *NIK'97, Norsk Informatikkonferanse*, pages 31–42. Tapir, November 1997.

Michael R. Blaha, William J. Premerlani, and James E. Rumbaugh. Relational database design using an object-oriented methodology. *Communications of the ACM*, 31(4):414–427, April 1988.

Grady Booch. *Object Oriented Design with Applications*. The Benjamin/Cummins Publ. Comp., 2 edition, 1994.

A. Borgida, S. Greenspan, and J. Mylopoulos. Knowledge representation as the basis for requirements specification. *IEEE Computer*, pages 82–91, 1985.

A.W. Brown and K.C. Wallnau. The current state of CBSE. *IEEE Software*, 15(5):37–46, September-October 1998.

J.F.M. Burg and R. P. van de Riet. Color-x: Using knowledge from wordnet for conceptual modeling. In Fellbaum (1998c), chapter 15, pages 353–377.

Mark Burstein. Combining analogies in mental models. In D.H. Helman, editor, *Analogical Reasoning*, pages 179–203. Kluwer Academic Publishers, 1988.

Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal. *Pattern-Oriented Software Architecture - A System of Patterns*. Wiley and Sons, 1996. ISBN 0471958697.

J.G. Carbonell. Learning by analogy: Formulating and generalizing plans from past experience. In R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, pages 137–161. Tioga, Palo Alto, CA, 1983a.

J.G. Carbonell. Derivational analogy and its role in problem solving. In *Proceedings AAAI-83*, pages 64–69, Pittsburgh, PA, 1983b.

S. Castano and V. De Antonellis. A reconstructive approach to reuse of conceptual components. In Prieto-Díaz and Frakes (1993), pages 19–28.

Alejandra Cechich and Mario Piattini. Early detection of COTS component functional suitability. *Information and Software Technology*, 49:108–121, 2007. URL `http://dx.doi.org/doi:10.1016/j.infsof.2006.03.008`.

Roger Chaffin and Douglas J. Herrmann. The nature of semantic relations: a comparison of two approaches. In Martha Walton Evens, editor, *Relational models of the lexicon. Representing knowledge in semantic networks*, Studies in natural language processing, chapter 13, pages 289–334. Cambridge University Press, 1988.

P. Chaiyasut, G. Shanks, and P. M. C. Swatman. A computational architecture to support conseptual data model reuse by analogy. In *Seventh International Workshop on Computer-Aided Software Engineering*, pages 70–79, Toronto, Canada, 1995. IEEE Computer Society Press.

Chia-Chu Chiang and David Neubart. Constructing reusable specifications through analogy. In *SAC'99, Proc. of the 1999 ACM symposium on Applied computing*, pages 586–592, San Antonio, Texas, 1999.

Reidar Conradi. Process support for reuse. In *10th International Software Process Workshop (ISPW '96)*, pages 43–47. IEEE Computer Society, 1996.

Brad J. Cox. *Object Oriented Programming, An Evolutionary Approach*. Addison-Wesley Publishing Company, Reading, Massachusetts, 2 edition, 1987.

R. Creps, R. Prieto-Diaz, M. Davis, M. Simos, P. Collins, and G. Wickman. Using a conceptual framework for reuse processes as a basis for reuse adoption and planning. Technical report, December 1993. URL `citeseer.nj.nec.com/creps93using.html`.

Pádraig Cunningham, Donal Finn, and Seá Slattery. Knowledge engineering requirements in derivational analogy. In Stefan Wess, Klaus-Dieter Althoff, and Michael M. Richter, editors, *EWCBR93, 1st European Workshop on Topics in Case-Based Reasoning*, pages 234–245, Kaiserslautern, Germany, November 1993. Springer-Verlag.

L. Peter Deutch. Design reuse and frameworks in the Smalltalk-80 programming system. In Ted J. Biggerstaff and Alan J. Perlis, editors, *Software Reusability, Vol. II, Applications and Experience*, pages 55–71. Addison Wesley, 1989.

Barry Ellingsen. *Connectionist-Based Analogical Mapping*. PhD thesis, Dept. of Information Science, University of Bergen, 1997a.

Barry K. Ellingsen. Distributed representations for analogical mapping. 9th European Conf. on Machine Learning, pages 25–32, Prague, April 1997b.

ERCIM92. *ERCIM Workshop on Methods and Tools for Software Reuse*, Heraklion, Greece, October 29–November 1 1992.

T. G. Evans. A program for the solution of geometric analogy intelligence test questions. In M. Minsky, editor, *Semantic Information Processing*. MIT Press, Cambridge, 1968.

Geir Fagerhus and Even-André Karlsson. Organizing and managing reuse. Technical Report REBOOT 8206.4, Q-Labs, October 10 1992.

Brian Falkenhainer. The SME user's manual (SME version 2e). Technical Report UIUCDCS-R-88-1421, Department of Computer Science, University of Illiois at Urbana-Champaign, December 1988.

Brian Falkenhainer, Kenneth D. Forbus, and Dedre Gentner. The structure-mapping engine: Algorithm and examples. *Artificial Intelligence*, 41:1–63, 1989.

C. Fellbaum. English verbs as a semantic net. http://www.cogsci.princeton.edu/∼wn/, 1993.

Christiane Fellbaum. Introduction. In Fellbaum (1998c), pages 1–19.

Christiane Fellbaum. A semantic network of english verbs. In Fellbaum (1998c), chapter 3, pages 69–104.

Christiane Fellbaum, editor. *WordNet, an Electronic Lexical Database*. The MIT Press, Cambridge, USA, 1998c.

Eduardo B. Fernandez. Building systems using analysis patterns. In *ISAW3*, pages 37–40, Orlando, Florida, 1998.

C.J. Fillmore and B.T.S. Atkins. Towards a frame-based lexicon: The semantics of RISK and its neighbors. In A. Lehrer and E. Feder Kittay, editors, *Frames, fields, and contrasts*, pages 75–102. Erlbaum, Hillsdale, NJ, 1992.

Gene Forte and Ronald J. Norman. A self-assessment by the software engineering community. *Communications of the ACM*, 35(4):28–32, April 1992.

Gilles Fouqué and Stan Matwin. A case-based approach to software reuse. *Journal of Intelligent Information Systems*, 1:165–197, 1993.

Martin Fowler. *Patterns of Enterprise Application Architecture*. The Addison-Wesley Signature Series. Addison-Wesley, 2003.

Martin Fowler. *Analysis Patterns. Reuseable Object Models*. Addison-Wesley, 1997.

Martin Fowler. *Refactoring. Improving the Design of Existing Code*. Addison-Wesley, 1999.

William B. Frakes. Software engineering. In *AccessScience@McGraw-Hill*. McGraw-Hill, 2002. http://www.accessscience.com/server-java/Arknoid/science/AS/Encyclopedia/6/63/Est-_631400_frameset.html; accessed May 13, 2004.

Harald Gall, Mehdi Jazayeri, and René Klösh. Research directions in software reuse: Where to go from here? In Samadzadeh and Zand (1995), pages 225–228. Special Issue.

E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patters, Elements of Reusable Object-Oriented Software*. Professional Computing Series. Addison-Wesley, 1995.

Dedre Gentner. Structure mapping: A theoretical framework for analogy. *Cognitive Science*, 7(2): 155–170, 1983.

M. R. Girardi and B. Ibrahim. Automatic indexing of software artifacts. In *Advances in Software Reuse. Proc. of the Third International Conference on Software Reusability*, pages 24–32, Los Alamitos, CA, 1994.

Bjorn Gronquist, Claude Villermain, and Blandine Bongard. A component approach for reuse. In *Proc. TOULOUSE -92*, Toulouse, France, December5–7 1992. EC2.

Jon Atle Gulla, Bram van der Vos, and Ulrich Thiel. An abductive, linguistic approach to model retrieval. *Data & Knowledge Engineering*, 23:17—31, 1997.

Rogers P. Hall. Computational approaches to analogical reasoning: A comparative analysis. *Artificial Intelligence*, 39:39–120, 1989.

Terry Halpin. UML data models from an ORM perspective: Part 2. *Journal of Conceptual Modeling*, 1998. URL `http://www.orm.net/pdf/ICMArticle2.pdf`.

Mehdi T. Harandi. The role of analogy in software reuse. In Ed Deaton, K. M. George, Hal Berghel, and George Hedrick, editors, *Proceedings of the ACM/SIGAPP Symposium on Applied Computing*, pages 40–47, Indianapolis, IN, February 1993. ACM Press.

Peter Herzum and Oliver Sims. *Business Component Factory*. John Wiley & Sons, Inc., 2000.

Nadine Heumesser and Frank Houdek. Towards systematic recycling of systems requirements. In *Proceedings of the 25th International Conference on Software Engineering*, pages 512–519, Portland, Oregon, May 3–10 2003.

J.H. Holland, K.J Holyoak, R.E. Nisbett, and P. Thagard. *Induction: Processes of Inference, Learning, and Creativity*. MIT Press, Cambridge, MA, 1986.

Michael Jackson. *Problem Frames*. ACM Press Books, 2001.

Michael Jackson. Problem frames and software engineering. *Information and Software Technology*, 47 (14):903–912, November 2005.

Jun-Jang Jeng and Betty H. C. Cheng. Using analogy and formal methods for software reuse. In *Proc. of IEEE 5th International Conference on Tools with AI*, pages 113–116, 1993.

Jun-Jang Jeng and Betty H. C. Cheng. A formal approach to reusing more general components. In *Proc. of IEEE 9th Konwledge-Based Software Engineering Conference*, September 1994.

Jun-Jang Jeng and Betty H. C. Cheng. Specification matching for software reuse: A foundation. In Samadzadeh and Zand (1995), pages 97–105. Special Issue.

Ralph E. Johnson and Brian Foote. Designing reusable classes. *Journal of Object-Oriented Programming*, 1(2):22–35, June/July 1988.

E.-A. Karlsson, G. Sindre, S. Sørumsgård, and E. Tryggeseth. Weighted term spaces for relaxed search. In *Proc. CIKM-92*, Baltimore, USA, November 8–11 1992. ISMM.

Smadar Kedar-Cabelli. Analogy—from a unified perspective. In D.H. Helman, editor, *Analogical Reasoning*, pages 65–103. Kluwer Academic Publishers, 1988.

Elizabeth A. Kendall. Role model designs and implementations with aspect-oriented programming. In *Proceedings of the Conference on Object-oriented Programming Systems, Languages and Applications*, pages 353–369, Denver, CO, USA, November 1999. Association for Computing Machinery, ACM Press.

R. E. Kling. A paradigm for reasoning by analogy. *Artificial Intelligence*, 2(2):147–178, 1971.

Cris Kobryn. Modeling components and frameworks with UML. *Communications of the ACM*, 43(10): 31–38, 2000.

Janet L. Kolodner. Reconstructive memory: A computer model. *Cognitive Science*, 7:281–328, 1983.

J.L. Kolodner, R.L. Simpson, and K. Sycara-Cyranski. A process model of case-based reasoning in problem solving. In *Proceedings IJCAI-9*, pages 284–290, Los Angeles, CA, 1985.

John R. Koza. *Genetic Programming II: Automatic Discovery of Reusable Progrms*. The MIT Press, 1998.

Glenn E. Krasner and Stephen T. Pope. A cookbook for using the model-view-controller user interface paradigm in smalltalk-80. Technical report, ParcPlace Systems, 2400 Geng Road Palo Alto, CA 94303, October 1987. Draft.

W. Lam, S. Jones, and C. Britton. Technology transfer for reuse: a management model and process improvement framework. In *Proc. of the Third International Conference on Requirements Engineering*, pages 233–240, 1998.

Hing-Yan Lee. *Automated Acquisition and Refinement of Reusable Software Design Components*. PhD thesis, University of Illinois at Urbana-Champaign, 1992.

Hing-Yan Lee and Mehdi T. Harandi. An analogy-based retrieval mechanism for software design reuse. In *The 8th Knowledge-Based Software Engineering Conference*, pages 152–159. IEEE, September 1993.

John A. Lewis. An experiment to determine software reusability factors. In *Proc. of the 1990 ACM annual conference on Cooperation*, page 405, Washington, USA, 1990.

George F Luger. *Artificial Intelligence, Structures and Strategies for Complex Problem Solving*. Addison Wesley, 4 edition, 2002.

Chung-Horng Lung and Joseph E. Urban. Integration of domain analysis and analogical approach for software reuse. In *ACM SAC'93*, pages 48–53, IN, USA, 1993.

Chung-Horng Lung and Joseph E. Urban. An approach to the classification of domain models in support of analogical reuse. In Samadzadeh and Zand (1995), pages 169–178. Special Issue.

Chung-Horng Lung and Joseph E. Urban. An expanded view of domain modeling for software analogy. To appear in Compsac, Aug. 1995, 1995b.

Chung-Horng Lung, Joseph E. Urban, and Gerald T. Mackulak. Analogy-based domain analysis approach to software reuse. *Requirements Engineering*, 12(1):1–22, 2007. DOI: 10.1007/s00766-006-0035-8.

P. Maguire, V. Shankararaman, R. Szegfue, and L. Morss. Application of case-based reasoning (CBR) to software reuse. *Lecture Notes in Computer Science*, 1020:166–??, 1995. ISSN 0302-9743.

N.A. Maiden and A.G. Sutcliffe. Exploiting reusable specifications through analogy. *Communications of the ACM*, 35(4):55–64, 1992.

N.A.M. Maiden and A.G. Sutcliffe. Computational mechanisms for reuse of domain knowledge during requirements engineering. Technical Report NATURE-94-08, Centre for Human-Computer Interface Design, City University, London, UK, 1994.

N.A.M. Maiden and A.G. Sutcliffe. Analogical retrieval in reuse-oriented requirements engineering. *Software Engineering Journal*, pages 281–292, September 1996.

Neil A. Maiden. *Analogical Specification Reuse During Requirements Analysis*. PhD thesis, School of Informatics, City University, London, 1992.

Neil A. Maiden and Alistair G. Sutcliffe. Requirements engineering by example: An empirical study. In *Proceedings of First International Symposium on Requirements Engineering*, pages 104–112. IEEE Computer Society Press, 1993.

Frank McCarey, Mel Ó Cinnéide, and Nicholas Kushmerick. Rascal: A recommender agent for agile reuse. *Artificial Intelligence Review*, 24(3-4):253–276, 2005.

M. D. McIllroy. Mass-produced software components. In *Software Engineering Concepts and Techniques, 1968. NATO Conference on Software Engineering*, pages 88–98, 1976.

Jean L. McKechnie, editor. *Webster's New Twentieth Century Dictionary Unabridged*. Simon and Schuster, 2 edition, 1979.

Dieter Merkl, A Min Tjoa, and Gerti Kappel. Learning the semantic similarity of reusable software components. In *Proc. of the Third International Conference on Software Reuse: Advances in Software Reusability*, pages 33–41, 1994.

Bertrand Meyer. Reusability: The case for object-oriented design. *IEEE Software*, 4(2):50–64, March 1987.

Bertrand Meyer. Applying “Design by contract”. *IEEE Computer*, 25(10):40–51, 1992.

Kjetil Midttun. Analogical retrieval in a repository of analysis specifications. Master’s thesis, Dept. of Information Science, University of Bergen, 1998. in Norwegian.

G. A. Miller, R. Beckwith, C. Fellbaum, D. Gross, and K. Miller. Introduction to WordNet. URL=http://www.cogsci.princeton.edu/∼wn/, August 1993.

George A. Miller. Nouns in WordNet. In Fellbaum (1998c), chapter 1, pages 23–46.

George A. Miller. Personal communication, July 2000.

Katherine Miller. Modifiers in WordNet. In Fellbaum (1998c), chapter 2, pages 47–67.

James M. Neighbors. Draco: A method for engineering reusable software systems. In Biggerstaff and Perlis (1989), chapter 12, pages 295–319.

Eduardo Ostertag, James Hendler, Rubéen Prieto Díaz, and Christine Braun. Computing similarity in a reuse library system: An AI-based approach. *ACM Transactions of Software Engineering and Methodology*, 1(3):205–228, July 1992.

Stephen Owen. *Analogy for Automated Reasoning*. Academic Press, Harcourt Brace JOvanovich, London, 1990.

D.L. Parnas. Software aging. In *Proc. of the 16th International Conference on Software Engineering*, pages 279–287, 1994. ICSE-16.

X. Pintado and D. Tsichritzis. An affinity browser. In D. Tsichritzis, editor, *Active Object Environments*. Centre Universitaire d’Informatique, Université de Genève, Geneva, 1988.

Bruce W. Porter, Ray Bareiss, and Robert C. Holte. Concept learning and heuristic classification in weak-theory domains. *Artificial Intelligence*, 45:229–263, 1990.

Jeffrey S. Poulin. Reuse: Been there, done that. *Communications of the ACM*, 42(5):98–100, 1999.

Rubén Prieto-Díaz. *A Software Classification Scheme*. PhD thesis, University of California, Irvine, 1985.

Rubén Prieto-Díaz and William B. Frakes, editors. Lucca, Italy, March 24–26 1993. IEEE CS Press.

Trygve Reenskaug, Per Wold, and Odd Arild Lehne. *Working With Objects. The OOram Software Engineering Method*. Manning Publications Co., 1996.

Danielle Ribot, Blandine Bongard, and Claude Villermain. Development life-cycle WITH reuse. In *Proceedings of the ACM 9th Symposium on Applied Computing*, Phoenix, USA, March 6–8 1994. Also available as WithReuseMethod-sac94.ps.Z from the REBOOT anonymous ftp-site.

R. Richardson and A. F. Smeaton. Using wordnet in a knowledge-based approach to information retrieval. Technical Report CA-0395, Dublin City University, Glasnevin, Dublin, 1995.

Dirk Riehle. *Framework Design. A Role Modeling Approach*. PhD thesis, ETH Züich, 1999. Dissertation No. 13509.

Linda Rising, editor. *The Patterns Handbook*. Cambridge University Press, 1998.

Mary Beth Rosson and John M. Carrol. Climbing the smalltalk mountain. *SIGCHI Bulletin*, 21, January 1990.

Mansur Samadzadeh and Mansour Zand, editors. *Proceedings of the Symposium on Software Reusability, SSR'95*, Seattle, WA, April 1995. ACM Press. Special Issue.

Douglas C. Schmidt. Why software reuse has failed and how to make it work for you. *C++ Report*, January 1999.

Douglas C. Schmidt. Using design patterns to develop reusable object-oriented communication software. *Communications of the ACM*, 38(10):65–74, October 1995.

Glenn Shafer. *A Mathematical Theory of Evidence*. Princteon University Press, Princeton, New Jersey, 1976.

Karma Sherif and Ajay Vinze. Domain engineering for developing software repositories: a case study. *Decision Support Systems*, 33(1):55–69, May 2002.

Karma Sherif, Radha Appan, and Zhangxi Lin. Resources and incentives for the adoption of systematic software reuse. *International Journal of Information Management*, 26(1):70–80, February 2006.

R.L. Simpson Jr. *A Computer Model of Case-Based Reasoning in Problem Solving: An Investigation in the Domain of Dispute Mediation*. PhD thesis, Georgia Institute of Technology, 1985.

Guttorm Sindre and Sivert Sørumgård. Terminology evolution in component libraries. In *Proc. 3rd International Congress on Terminology and Knowledge Engineering*, Cologne, Germany, Aug 24–27 1993.

Ian Sommerville. *Software Engineering*. Addison-Wesley, 7 edition, 2004.

L. S. Sørumgård, G. Sindre, and F. Stokke. Experiences from application of a faceted classification scheme. In Prieto-Díaz and Frakes (1993).

L.S. Sørumgård, G. Sindre, and F. Stokke. Experiences in reusable component classification. In ERCIM92 ERCIM92.

John F. Sowa, editor. *Principles of Semantic Networks, Explorations in the Representation of Knowledge*. Morgan Kaufmann, San Mateo, California, 1991.

George Spanoudakis and Panos Constantopoulos. Similarity for analogical software reuse: A conceptual modelling approach. In ERCIM92 ERCIM92. URL `Also available from ftp.informatik.twth-aachen.de as NATURE-92-8.ps.Z`.

George E. Spanoudakis and Panos Constantopoulos. Analogical reuse of requirements specifications: A computational model. *Applied Artificial Intelligence*, 10(4):281–306, 1996a.

George E. Spanoudakis and Panos Constantopoulos. Elaborating analogies from conceptual models. *International Journal of Intelligent Systems*, 11(11):917–974, 1996b.

Gorm Erik Steine-Eriksen. ROSA-repository. the development of an object-oriented CASE-repository. Master's thesis, Dept. of Information Science, University of Bergen, October 1995. in Norwegian.

Alistair Sutcliffe. Domain analysis for software reuse. *Journal of Systems and Software*, 50(3):175–199, 2000.

Gilbert Syswerda. A study of reproduction in generational and steady state genetic algorithms. In Gregory J. E. Rawlings, editor, *Foundations of genetic algorithms*, pages 94–101. Morgan Kaufmann, San Mateo, 1991.

Clemens Szyperski, Dominik Gruntz, and Stephan Murer. *Component Software. Beyond Object-Oriented Programming*. Addison-Wesley, 2 edition, 2002.

B. Tessem, S. Bjørnestad, K.M. Tornes, and G. Steine-Eriksen. ROSA = Reuse of Object-oriented Specifications through Analogy: A project framework. Technical Report No. 16, ISSN 0803–6489, Dept. of Information Science, Univ. of Bergen, 1994.

Bjørnar Tessem. Structure abstractions in retrieval of analogical software models. *Expert Systems with Applications*, 15:341–348, 1998a.

Bjørnar Tessem. Genetic algorithms for analogical mapping. In *Proc. of Intl. Conf. on Evolutionary Computation (ICEC'98)*, pages 762–777, Anchorage, Alaska, 1998b. IEEE.

Bjørnar Tessem and Solveig Bjørnestad. Analogy and complex software modeling. *Computers in Human Behavior*, 13(4):465–486, 1997.

Paul Thagard. Dimensions of analogy. In D.H. Helman, editor, *Analogical Reasoning*, pages 105–124. Kluwer Academic Publishers, 1988.

Knut Martin Tornes. The ROSA modeling tool. Master's thesis, Dept. of Information Science, University of Bergen, 1995.

Will Tracz. Personal email, April 2004.

Will Tracz, editor. *Software Reuse: Emerging Technology*. IEEE Computer Society Press, Los Alamitos, CA, 1988. Tutorial, Part 1, Overview.

Will Tracz. Software reuse myths revisited. In *Proc. of the 16th International Conference on Software Engineering*, pages 271–272, 1994a.

Will Tracz. Reuse state of the art and state of the practice report card. In *Proc. of the 3rd International Conference of Software Reuse: Advances in Software Reusability*, pages 193–194, 1994b.

Will Tracz. Confessions of a used-program salesman: Lessons learned. In Samadzadeh and Zand (1995), pages 11–13. Special Issue.

Ahsan Ul-Haq. An analogy machine for ROSA, adapting sme to ooram analysis models. Master's thesis, Dept. of Information Science, University of Bergen, Norway, November 1997.

B. C. Vickery. *Faceted Classification: A Guide to Construction and use of Special Schemes*. Aslib, 3 Belgrave Square, London, 1960.

Matthew Wall. Galib documentation. http://lancet.mit.edu/galib-2.4/, 1996. Version 2.4.2.

Kurt Wallnau, Scott Hissam, and Robert Seacord. *Building Systems from Commercial Components*. Addison Wesley, 2002. ISBN ISBN: 0201700646. URL `http://www.aw.com/catalog/academic/product/1,4096,0201700646,00.html`.

R. Alan Whitehurst. *Systematic Software Reuse Through Analogical Reasoning*. PhD thesis, University of Illinois, Urbana-Champaign, IL, 1995.

Frank Wilcoxon and Roberta A. Wilcox. Some rapid approximate statistical procedures. Technical report, Lederle Laboratories, Perl River, NY, 1964.

M.E. Winston, R. Chaffin, and D.J. Hermann. A taxonomy of part-whole relations. *Cognitive Science*, 1987.

Patrick Henry Winston. Learning and reasoning by analogy. *Communications of the ACM*, 23(12): 689–702, 1980.

Patrick Henry Winston. Learning new principles from precedents and exercises. *Artificial Intelligence*, 19(3):321–350, 1982.

C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén. *Experimentation in Software Engineering. An introduction*. Kluwer Academic Publishers, 2000.

Mansour Zand, Vic Basili, Ira Baxter, Martin Griss, Even-Andre Karlsson, and Dewayne Perry. Reuse rd: Gap between theory and practice. In *Proceedings of the fifth symposium on Software Reusability*, pages 172–177, Los Angeles, CA, USA, May 1999. ACM Press.

Appendix A

The OOram Methodology

Reenskaug et al. (1996) define *object orientation* as the "modelling of interesting phenomena as structures of interacting objects". They claim that objects should always be considered in the context of their collaboration. While other definitions are directed more towards properties such as inheritance and polymorphism, Reenskaug et al. say that, although these properties are important, they are not the essence of object orientation. Since collaboration in object-oriented systems is emphasized, there is a need for a modelling technique that focuses on this aspect instead of the static properties, such as *classes* and inheritance relationships. Classes are seen as powerful implementation abstractions.

Role Models in OOram

OOram focuses on the *role model* as the basic object abstraction. "A real world phenomenon is described as a number of cooperating objects. Sub-phenomena are specified by their area of concern, objects describing a sub-phenomenon are organized in a pattern of objects, and all objects having the same position in the pattern are represented by a role. We say they play the same role in the object structure" (Reenskaug et al., 1996, p. 7).

A *role* has identity and is encapsulated (like an object), and it is an archetypical representation of the objects occupying the corresponding positions in the object system. A *role model* describes the subject of the object interaction, the relationships between objects, the messages that each object may send to its collaborators, and the model information processes. It describes how a structure of objects achieves a given area of concern. According to these definitions an object may play different roles, and one role may be played by several types, or classes, of objects. This allows us to treat different aspects of an object, or object tasks, in separate role models. OOram supports the following *views* of role models as

- *role model collaboration view*—shows the roles, their attributes and their collaboration structure
- *interface view*—shows the message interfaces
- *scenario view*—shows the message flow between roles, such as UML *sequence diagrams*
- *method view*—shows what happens in a role when it receives a message, i.e., when it is triggered.

It is up to a user of the methodology to choose which views are suitable for a particular project. Each *role model* consists of a set of roles, where the *stimulus role* initiates the activity in the role model. If a role in the model can communicate with other roles in the model, there must be a path between the two roles. Each role can have a set of ports, and each port is linked to another role through a path. Each port can define a set of messages. Messages may be sent over the path. This set of messages of a port represents the interface to the role at the far end of the path. The role at one end of a path knows about zero, one, or many roles at the other end of the path. If it has knowledge of no roles, there is no port. An example role model is given in figure A.1. Here the role *borrower* is the stimulus role. A role model can be a synthesization of several role models. Role models have knowledge about both their super- and sub-models.

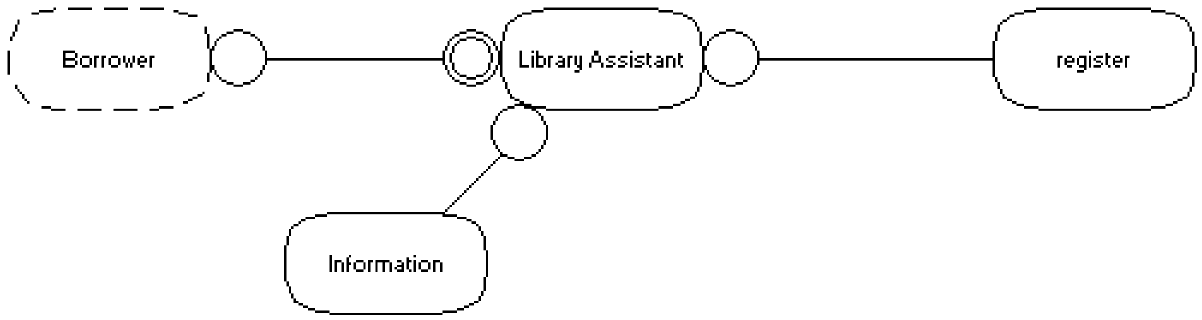


Figure A.1: The OOram role model Request library card.

Role Model Synthesis

Since one complex problem is described as a set of role models, each representing one area of concern, we need a way to combine these models to get an overview of all the roles of the problem. This is done by a process called *synthesis*. The term *role model synthesis* is used to denote the construction of a derived model from one or more base models in a controlled manner. Reenskaug et al. (1996) claim that while all object-oriented methods support inheritance, where a derived class can be defined to inherit attributes and behaviour from one or more base classes, in OOram, inheritance is lifted to the model level, so that a derived model inherits the static and dynamic properties of one or more base models.

Synthesis is used for three different purposes: 1) to create specialisation/generalisation relationships between models, 2) to compose a synthesized model from one or more base models on the same level, and 3) as an aggregation. Andersen and Reenskaug (1992) give an example of synthesis[1]. There are two important aspects of role model synthesis

- role models are combined vertically by letting their objects play multiple roles,
- integration of role models is done through the methods that objects use to process incoming messages.

Dependencies among synthesized role models are expressed in the methods of the objects. The behaviour of an object when receiving a message in the context of a role in one role model may be modified because it also plays some other role from another role model.

Implementation of Role Models

During implementation, the *object specification model* is introduced. It is the role model of a structure of objects that are implemented. An *object type* is a role in an object specification model, i.e., it is the specification of a set of objects with identical, externally observable properties. A *class* is an implementation of an object type.

Different classes can implement the same type. This is often done in an inheritance hierarchy, but is not essential. In Java it can be through an Interface. An object can play

[1]The notation used in their paper as well as several other prior papers is slightly different from that used in Reenskaug et al. (1996)

several roles, and these roles are then implemented in the same class. There exist many-to-many relationships between objects, roles, types, and classes as shown in figure A.2.

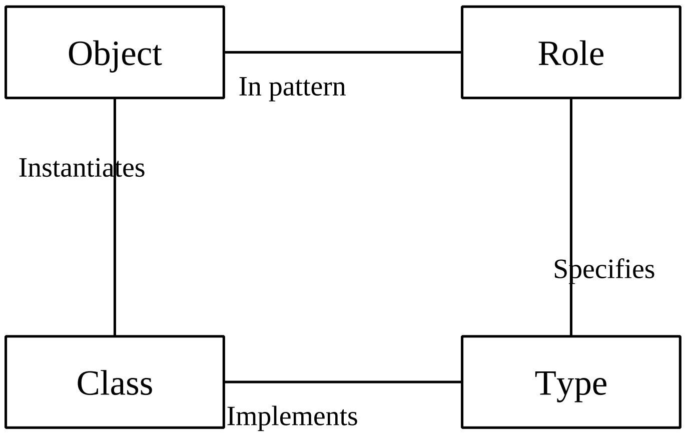


Figure A.2: Relationships among object, role, type, and class in OOram (from Reenskaug et al. (1996))

1. The object is the *is* abstraction, and represents a part of the system. It has identity and attributes and is encapsulated so that the messages it sends and receives constitute all its externally observable properties.
2. The role is the *why* abstraction. All objects serving the same purpose in a structure of collaborating objects, as viewed from the context of some *area of concern*, are said to play the same role.
3. The type is the *what* abstraction. All objects that exhibit the same externally observable properties are considered to belong to the same type.
4. The class is the *how* abstraction. All objects that share a common implementation are considered to belong to the same class.

Reuse in OOram

Reuse has always been considered important in OOram, and it has been emphasized that "reuse of ideas, design, and code" should be maximized. Reuse may be done at five levels during software development

1. *system user model*—composed from general patterns created in the current project or found in a repository

2. *system requirements model*—composed from general patterns created as part of the project or found in a repository
3. *system design model*—based on a number of patterns or frameworks from the repository
4. *system implementation*—derived from one or more framework classes in the repository
5. *system of objects*—composed from predefined library objects

Object-oriented code reuse is based on *inheritance* and *encapsulation*. This gives the objects similarity that can be searched for, protection from misuse, and the possibility to be replaced with another object with a similar behaviour. The last point is similar to the rule recommended in literature related to Java: “Program towards interfaces, not classes”. However, there seems to be no support for reuse in the OOram tool made by Taskon.

Appendix B

The WordNet Lexical Database

WordNet is a lexical database that supports conceptual, rather than merely alphabetical, search (Fellbaum, 1998c). It combines features from both a dictionary and a thesaurus. Contrary to many similar dictionaries with a restricted vocabulary, WordNet contains the full range of common English vocabulary. It is organized around meanings in synonym sets, or *synsets*, consisting of all the words that express a common concept, and with relationships between these synsets.

Each *word form*, i.e., the written word, can have several *word meanings*. Word forms belong to one of the word categories nouns, verbs, adjectives, or adverbs. Of the word categories, the noun group is by far the largest. The semantic relationships between synsets (examples below are taken from Miller et al. (1993)) include

- hyponymy—specialization, e.g., {*tree*} is a hyponym of {*plant*}, and {*plant*} is a hypernym of {*tree*}, i.e., a generalization,
- meronymy—part-whole relation, from Greek *meros* (part), e.g., a *car* has an *air bag*. The opposite of meronymy is holonymy, e.g., *air bag* is part of a *car*.
- entailment—implication, i.e., one statement implies others, e.g., the relation between the sentence *John is a bachelor* and the sentences *John is a male / John is an adult / John is unmarried*. These entailment relations reflect the fact that the lexical composition of *bachelor* includes the components *male*, *adult*, and *unmarried*. Entailment is only used between verbs (see section B.3).

Lexical relations can also exist between words rather than between synsets, e.g., antonymy, which expresses degree of oppositeness between words. This relation exists between nouns, e.g., *girl/boy*, between adjectives, e.g., *good/bad*, and between verbs, e.g., *appear/disappear*. These are relations between specific *word meanings*. Although opposite meanings can be identified among words other than direct antonyms, people are not likely to use them together in pairs. While *fall/rise* and *ascend/descend* are direct antonyms, *rise/descend* and *ascend/fall* are conceptually opposed, but they are not direct antonyms.

WordNet does not contain organizational units larger than words, but the relational semantics does reflect some of the structure of frame semantics, e.g., WordNet does relate words like *buy* and *sell* through the antonym relation, as in a frame for a *commercial transaction*; see Fillmore and Atkins (1992). According to Fellbaum (1998a), WordNet "relates words and concepts from a common semantic domain". The word categories are discussed separately in this chapter.

The basic relation used in WordNet is *synonymy*. Words are organized in synonym sets, or synsets for short. This implies that the words in a synset are interchangeable in some, although not in all, contexts. Some short phrases, or collocations, that would normally not be included in a dictionary, are included in WordNet.

B.1 Nouns

The most important relationship *between* noun synsets is *hyponymy* (subordination or specialization); see example in the list above. Hyponomy, according to Miller (1998a), organizes synsets into a set of lexical hierarchies, each with a *unique beginner* at its root. He describes 25 such beginners shown in table B.1. The features that characterize a unique beginner are inherited by all its hyponyms. A unique beginner can be regarded as a primitive semantic component of all words in its hierarchically structured semantic field.

The unique beginners are organized in 11 hierarchies, as they can be divided into natural groups according to what they are concerned about. These hierarchies are shown in figure B.1. The figure shows how the other beginners are organized within the 11 hierarchies. In order to create these hierarchies, a few new general synsets are introduced.

The hierarchies tend to be fairly shallow, and lexical inheritance systems are seldom more than 10–12 levels deep. The hierarchies in WordNet vary much both in terms of the

{*act, activity*}	{*food*}	{*possession*}
{*animal, fauna*}	{*group, collection*}	{*process*}
{*artifact*}	{*location*}	{*quantity, amount*}
{*attribute*}	{*motivation, motive*}	{*relation*}
{*body*}	{*natural object*}	{*shape*}
{*cognition, knowledge*}	{*natural phenomenon*}	{*state*}
{*communication*}	{*person, human being*}	{*substance*}
{*event, happening*}	{*plant, flora*}	{*time*}
{*feeling, emotion*}		

Table B.1: List of 25 unique beginners for WordNet nouns

number of synsets belonging to them and their depth. A noun has typically only a single hypernym, although there are examples in the WordNet database where this is not the case.

Miller distinguishes between the hyponymic relation types: IS-A-KIND-OF relation discussed before, and IS-USED-AS-A-KIND-OF (Miller, 1998a). While these may be important in natural language, it is unclear whether they are significant for system development.

A word's antonym is often the first word that comes to mind when a person hears a word. For example, given the word *man*, a person would most likely respond *woman*. This is a lexical, not a semantic, relation, and it is represented between *word meanings* and not synsets. If an antonym is defined between two words, this implies that there is some kind of weaker opposite relation between the other words in the two synsets as well.

Noun attributes are represented by the meronymy relation. Winston, Chaffin, and Hermann (1987) have identified 6 different meronymy relations. In WordNet they have coded three of these: COMPONENT-PART, e.g., *branch/tree*, MEMBER-OF, e.g., *tree/forest*, and the STUFF SOMETHING IS MADE OF, e.g., *aluminum/airplane*. A noun can have attributes that are described by adjectives (see section B.2.1). This kind of relation is not added to WordNet. The only way adjectives are used in relation to nouns is as modifiers, e.g., an *easy* chair, *straight* chair. These collocations are typically hyponyms of the noun.

B.2 Modifiers

Adjectives are used to modify nouns, and adverbs modify verbs, adjectives, other adverbs, and entire clauses or sentences (Miller, 1998b).

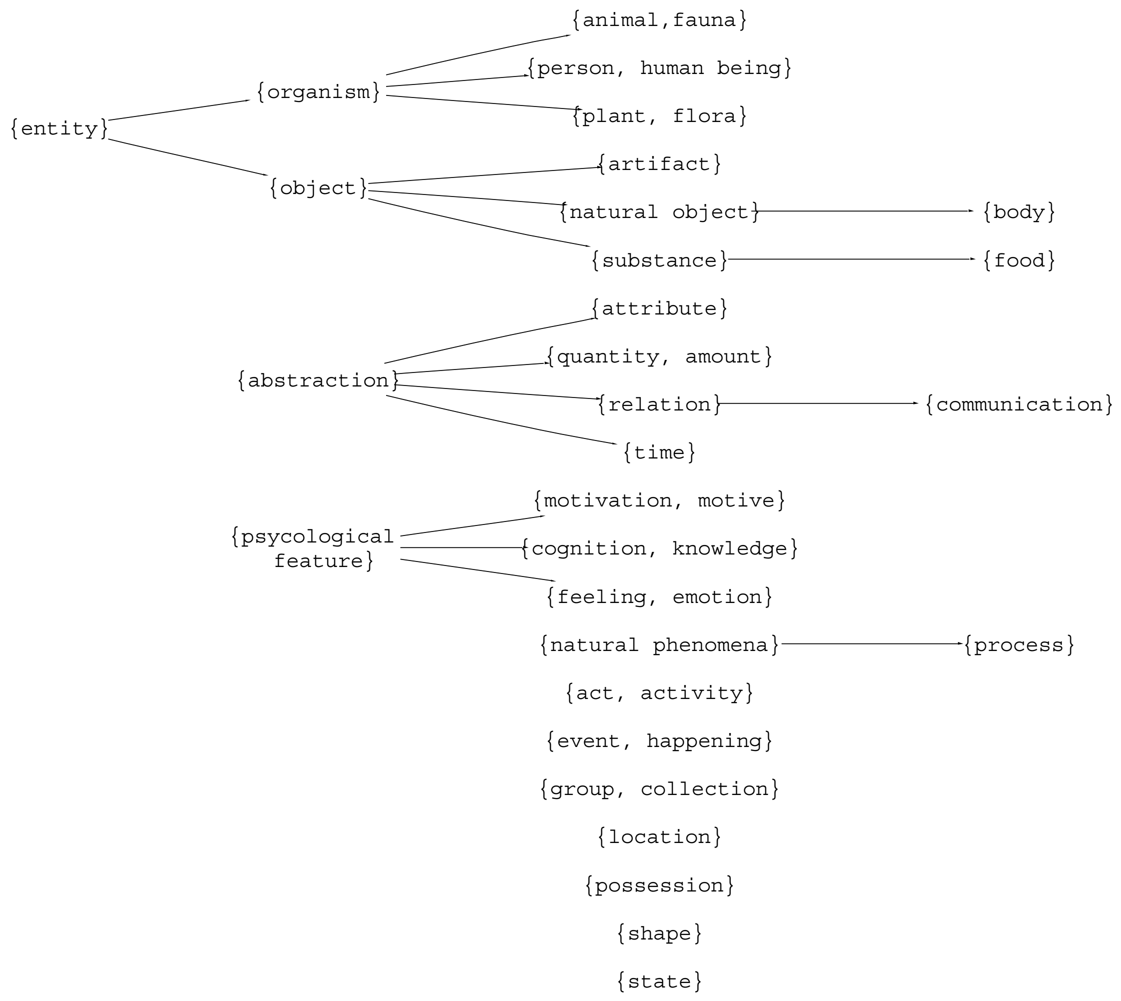


Figure B.1: The 11 noun hierarchies in WordNet

B.2.1 Adjectives

Adjectives are used to modify or elaborate on the meaning of nouns. In English, nouns can also be used as adjectives. In addition, prepositional phrases, and even entire clauses, may be used to modify nouns. WordNet divides adjectives in two broad categories. These are the *descriptive* adjectives, e.g., big, interesting, and possible, and *relational* adjectives, e.g., presidential, and nuclear. The two groups of adjectives are discussed separately.

Descriptive Adjectives

This is what one usually thinks of when adjectives are mentioned. We can say that they ascribe to a noun a value of an attribute. When we say that *a room is little*, *little* can be thought of as being the value of an attribute *size*. WordNet contains pointers between descriptive adjectives and the nouns by which appropriate adjectives are lexicalized. If you look up *little*, and select *little is a value of, size* is shown. *Little* is also shown as the opposite of *big*, i.e., its antonym.

Descriptive adjectives are organized differently from the other major categories, as antonymy is the basic semantic relation among them. Semantic similarity is introduced among synonym sets. An adjective A_1 is similar to an adjective A_2, and A_2 is antonym to an adjective A_3 as in "*Prompt* is similar to *fast* is antonymous to *slow*". For adjectives with no obvious antonym, the strategy has been to find a similar adjective that does have an antonym, and to code it as similar to one in that pair. Thus, adjectives form clusters around pairs of antonyms, as with the pair *fast/slow* which denotes the attribute speed. Each of these two adjectives forms the heads of a half cluster. The elements in these clusters are called satellite synsets. The satellites can be considered specializations of the head synset.

In WordNet, different senses of an ambiguous word frequently have different antonyms, e.g., *fresh/stale* bread, while *fresh/dirty* shirt. Adjectives are selective about the nouns they modify. The adjectives most widely used are those expressing evaluations, e.g., *good/bad*. Those expressing activity also have a wide applicability, e.g., *active/passive, fast/slow*. As the noun is considered more important for the meaning of an expression than the adjective used with it, the noun is given meaning first, and then the adjective is given meaning in relation to the noun.

Relational Adjectives

Relational adjectives are related semantically to nouns. Such adjectives play a similar role as a modifying noun and function as classifiers. An example is *musical instrument*, which is related to music. There is also a semantic relation between the adjective and the noun that it modifies, e.g., in *agricultural equipment* and *agricultural college*, agricultural has different relations to the noun. Relational adjectives do not refer to a property of the noun they modify. Few of them have a direct antonym.

B.2.2 Adverbs

Most adverbs are created from adjectives by adding a suffix. Most are derived by adding the suffix -*ly* to specify manner or degree. In WordNet, derived adverbs are linked to the adjective by the relation DERIVED-FROM. This link is by way of a particular adjective sense. The ones that are related to adjectives often also inherit the antonym relation. Adverbs with the form *adjective* + *-ly* are not semantically related to the adjective at all, e.g., *hardly*. WordNet also contains some phrases that are used adverbally.

Adverbs are organized by their relation to their adjective form. Moreover, only synonymy and sometimes antonymy is stored. Each synset contains one adverb, its related adjective, if any, any synonyms or antonyms, and a parenthetical gloss including example phrases.

B.3 Verbs

Verbs are, according to Fellbaum (1998b), organized as a network of related synsets basically in the same way as nouns. Verbs are more polysemous than nouns, i.e, they can take on more meanings. The higher polysemy suggests that verb meanings are more flexible than noun meanings. They can change their meanings depending on the noun arguments with which they co-occur. It is therefore more unclear what should constitute unique beginners for verb hierarchies.

In WordNet verbs are divided in 15 groups or semantic domains, although Fellbaum admits that the borders between these groups are often vague. Several suggestions for unique beginner verbs have been put forward, such as those described for nouns (see section B.1), and there is no general agreement concerning how to do this. One distinction may go between *translational movement* and *movement without displacement* (Fellbaum, 1998b). The motivation behind selection of unique beginner verbs is outside the scope of this thesis. I will emphasize, however, that they are, among other things, based on studies of how people seem to organize their mental lexicon, studies of the use of substitution errors, as well as studies of traditional dictionaries.

Although verbs are organized as synonym sets, there are few true synonyms among English verbs, one example being the verbs *close* and *shut*. Words that will typically not be used in the same context, e.g., as one represents a more formal context, are in WordNet organized in the same synset, e.g., *begin* and *commence*. To clarify the difference in use, glosses are used to show in what contexts the different members of a synset can be used.

B.3.1 Lexical and semantic relations

Relations used between verb synsets can be categorized by lexical entailment. This is a relation between two verbs, V_1 and V_2, that holds when the sentence *Someone* V_1 logically implies *Someone* V_2. This is a unilateral relation. If two verbs were mutually entailing, they must be synonyms. Negation reverses the entailment. Entailment is somehow similar to meronymy among nouns. An activity or event is part of another activity only when it is part of, or a stage in, its temporal realization. They can be occurring at the same time, like *drive* and *ride* (a car) or they can be temporally coexistent (at least partly) as in *snore* and *dream* which can be part of *sleeping*. Details about how one verb can entail the other, will not be discussed here.

Hyponymy among verbs is expressed differently than among nouns. We distinguish, for example, between ways things can move by the MANNER in which it moves. This manner relation is termed TROPONOMY by Fellbaum and Miller (referred in Fellbaum (1998b). Troponomy can be expressed as *To* V_1 *is to* V_2 *in some particular manner*. Manner can be thought of as the OCCASION for something to happen, e.g., fight, the INTENTION of the action or event, e.g., for communication verbs, or the MEDIUM of communication.

Troponomy is a special form of entailment. Every troponym V_1 of a more general verb V_2 also entails it. *March* is a troponym of *walk*, but *marching* entails *walking*. Activities referred to by a troponym and its more general superordinate are always temporarily coextensive. You must be *walking* if you are *marching*, but not the other way around. The other case, however, needs not be the case, i.e., an entailment needs not be a troponym. The example given before—*snore/sleep*—shows this. Verbs in pairs like this are related only by entailment and proper temporal inclusion, and they cannot also be related by troponomy.

The hierarchies of verbs tend to be more shallow and “bushy” than the noun hierarchies. They seldom exceed four levels. There also seems to be one level that contains much more verbs than the others. When looking at the taxonomy from one sense of talk, the more general level is *communicate*, the next lower level contains few verbs, including *talk*, *write*. The next level, however, has many troponyms, including *babble, mumble, slur*, etc. To say that *Talk is to communicate in some way* is totally acceptable, but to say that *To babble/mumble/slur . . . is to talk in some way* shows a closer resemblance. These words are obviously specific ways of talking. *Talk* seems to be more remote from its superordinate *communicate*. The further down the hierarchy we descend, the fewer nouns the verbs can take as a potential argument. This is natural since the verbs tend to take on a more and more specific meaning.

B.3.2 Polysemy

When identifying the senses for verbs in WordNet, the developers have split rather than lumped together senses. The most frequent used verbs (*have, be, run, make, set, go, take*, and others) are also the most polysemous, and their meanings depend heavily on the nouns with which they co-occur. For example, dictionaries differentiate between the senses of *have* in sentences such as *I have a BMW* and *I have a headache*. The difference is less due to the polysemy of *have*, however, than to the concrete or abstract nature of its objects. In WordNet they settled on more meaningful beginners.

B.4 Design of the WordNet Database

The WordNet lexical database is organized as a set of ASCII files that are converted to an indexed database file by a program calculating the offset into the file where a particular synset is found. The senses of a specific word form are ordered according to their frequency of use, with the most frequently used sense listed first. The frequencies are calculated based on a set of corpora. A word's familiarity is recorded in WordNet as a count of its polysemy. It has been found that this correlates well with people's familiarity with it.

For each word form, a number is stored that represents the polysemy for each word category in which it occurs. For each word category, a data file and an index file are created. The index file contains each word form, its polysemy count, and the offsets for the synsets that contain it. The sequence of the synset address represents the sense number. The relational information is given by the type of relation, in the form of a relational symbol, the address of the target synset, and its syntactic category. The last piece of information is necessary in cases where the target synset belongs to another syntactic category. The types of relations coded in WordNet for the different syntactical categories are listen in table B.2.

Noun	Verb	Adjective	Adverb
Antonym	Antonym	Antonym	Antonym
Hyponym	Troponym	Similar	Derived from
Hypernym	Hypernym	Relational adj	
Meronym	Entailment	Also see	
Holonym	Cause	Attribute	
Attribute	Also see	Participle	

Table B.2: Relations coded in WordNet

B.4.1 Morphology

In traditional dictionaries, typically only a head word is listed, with the various morphological forms being put in a sub-listing. This typically does not cause a problem. When searching for a word, however, it may cause a problem if the system just reports that the word does not exist in the database. In WordNet only the base forms are listed. To spare the user from stripping affixes, a program handles morphological transformations. Rules on how to handle regular cases are added, and exception lists are added to take care of the rest, one for each word category. Collocations are typically more complicated than single words, and verb phrases are more complicated than compound nouns, especially if they contain a preposition.

Appendix C

The DTD for OOram Models

Two versions of the DTD has been used in the experiments. The first version shown in C.1 is used in all experiments before introducing the modifiers in role names, while the second version, in C.2, is used in the examples thereafter. The only differences between them are related to the `mName`.

In XML, tags of the form < !- - some text - -> represents comments.

C.1 The Initial DTD

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- DTD for OOramModelCollection S. Bjoernestad 2000.05.02  -->
<!-- The DTD is used to read a file of unrelated OOram Models. -->
<!-- An OOramModelCollection may have one or more Role models -->

<!ELEMENT OOramModelColl  (RoleModel* )>

<!ELEMENT RoleModel  (Name , descr? , Project? ,
                      Developer? , modelLink? , Role+ , Port+ )>
<!ATTLIST RoleModel  id   ID  #REQUIRED
               kind (roleModel | objectSpecification | typeModel )
                     'roleModel'>
<!ELEMENT Project EMPTY>
<!ATTLIST Project  name CDATA  #REQUIRED >

<!ELEMENT Developer EMPTY>
<!ATTLIST Developer  logonName CDATA  'solveig' >

<!-- A Role has a rName, a description, and any number of ports. -->
<!ELEMENT Role  (rName, descr? , Port* , roleLink? )>
<!ATTLIST Role  id   ID  #REQUIRED
                kind  (stimulus | internal )  'internal' >
<!ELEMENT rName EMPTY>
<!ATTLIST rName  name     CDATA  #REQUIRED
                 synsetID CDATA  #IMPLIED >
```

```
<!-- A role model can be synthesized from several role models. -->
<!-- A role model is part of just one synthesized role models. -->
<!ELEMENT modelLink EMPTY>
<!ATTLIST modelLink  subModels IDREFS  #IMPLIED >

<!-- similar as role model links. -->
<!ELEMENT roleLink EMPTY>
<!ATTLIST roleLink  subRoles IDREFS  #IMPLIED >

<!-- A Port may have a set of Messages. -->
<!ELEMENT Port  (Message* )>

<!-- A Port must specify the from and to roles. -->
<!-- It can have an attribute card. 2 = N. -->
<!-- If not given, the default is 1 -->
<!-- It can also have a name, however, this is not recommended -->
<!ATTLIST Port fromRole CDATA  #REQUIRED>

<!ATTLIST Port toRole   CDATA  #REQUIRED>

<!ATTLIST Port card      (1 | 2 )  "1">

<!ATTLIST Port name     CDATA  #IMPLIED>

<!ELEMENT Message  (Name , Arg* , RetType? )>

<!ELEMENT Arg  ( (Name , Type )+ )>

<!ELEMENT Name  (#PCDATA )>

<!ELEMENT descr  (#PCDATA )>

<!ELEMENT RetType  (Type )>

<!ELEMENT Type  (#PCDATA )>
```

C.2 The Introduction of Modifiers

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- DTD for OOramModelCollection S. Bjoernestad 2002.05.20  -->
<!-- This version has introduced the use of modifiers in role names -->
<!-- The DTD is used to read a file of unrelated OOram Models. -->
<!-- An OOramModelCollection may have one or more Role models -->

<!ELEMENT OOramModelColl  (RoleModel* )>

<!ELEMENT RoleModel  (Name , descr? , Project? ,
                      Developer? , modelLink? , Role+ , Port+ )>
```

```
<!ATTLIST RoleModel  id   ID  #REQUIRED
               kind (roleModel | objectSpecification | typeModel )
                     'roleModel'>
<!ELEMENT Project EMPTY>
<!ATTLIST Project  name CDATA  #REQUIRED >

<!ELEMENT Developer EMPTY>
<!ATTLIST Developer  logonName CDATA  'solveig' >

<!-- A Role has a rName, may have mName + any number of ports. -->
<!ELEMENT Role  (rName , mName? , descr? , Port* , roleLink? )>
<!ATTLIST Role  id   ID  #REQUIRED
                kind  (stimulus | internal )  'internal' >
<!ELEMENT rName EMPTY>
<!ATTLIST rName  name     CDATA  #REQUIRED
                 synsetID CDATA  #IMPLIED >

<!ELEMENT mName EMPTY>
<!ATTLIST mName  name CDATA  #REQUIRED >

<!-- A role model can be synthesized from several role models. -->
<!-- A role model is part of just one synthesized role models. -->
<!ELEMENT modelLink EMPTY>
<!ATTLIST modelLink  subModels IDREFS  #IMPLIED >

<!-- similar as role model links. -->
<!ELEMENT roleLink EMPTY>
<!ATTLIST roleLink  subRoles IDREFS  #IMPLIED >

<!-- A Port may have a set of Messages. -->
<!ELEMENT Port  (Message* )>

<!-- A Port must specify the from and to roles. -->
<!-- It can have an attribute card. 2 = N. -->
<!-- If not given, the default is 1 -->
<!-- It can also have a name, however, this is not recommended -->
<!ATTLIST Port fromRole CDATA  #REQUIRED>

<!ATTLIST Port toRole   CDATA  #REQUIRED>

<!ATTLIST Port card      (1 | 2 )  "1">

<!ATTLIST Port name     CDATA  #IMPLIED>

<!ELEMENT Message  (Name , Arg* , RetType? )>

<!ELEMENT Arg  ( (Name , Type )+ )>
```

```
<!ELEMENT Name  (#PCDATA )>

<!ELEMENT descr  (#PCDATA )>

<!ELEMENT RetType  (Type )>

<!ELEMENT Type  (#PCDATA )>
```

Appendix D

Example XML OOram Role Model

```xml
<?xml version="1.0"?>
<!DOCTYPE OOramModelColl SYSTEM "OOramModelColl.dtd">
<!-- Created by XMLBuilder -->
 <OOramModelColl>
 <RoleModel id="m1">
  <Name>Request\_Library\_Card</Name>
  <Role id="r1" kind="stimulus">
    <rName name="borrower" synsetID="107121607"/>
    <descr>asks for a library card</descr>
  </Role>
  <Role id="r2">
    <rName name="assistant" synsetID="107240707"/>
    <mName name="library"/>
    <descr>helps the borrowers</descr>
  </Role>
  <Role id="r3">
    <rName name="information" synsetID="106250971"/>
    <descr>the record stored in the system</descr>
  </Role>
  <Role id="r4">
    <rName name="database"/>
    <mName name="reference"/>
  </Role>
  <Port fromRole="r1" toRole="r2"></Port>
  <Port fromRole="r2" toRole="r1" card="2"></Port>
  <Port fromRole="r2" toRole="r3"></Port>
  <Port fromRole="r2" toRole="r4"></Port>
 </RoleModel>
</OOramModelColl>
```

```xml
<?xml version="1.0" ?>
<!DOCTYPE OOramModelColl SYSTEM "OOramModelColl.dtd">
<!--  Created by XMLBuilder  -->
 <OOramModelColl>
 <RoleModel id="m1" kind="roleModel">
  <Name>Library_sub</Name>
  <descr>registers information about borrower</descr>
 <Role id="r1" kind="stimulus">
  <rName name="borrower" synsetID="107121607" />
  <descr>lends books</descr>
 </Role>
 <Role id="r2" kind="internal">
  <rName name="library" synsetID="102920588" />
  <descr>contain books</descr>
 </Role>
 <Role id="r3" kind="internal">
  <rName name="library assistant" synsetID="107240710" />
  <descr>helps the borrowers</descr>
 </Role>
 <Role id="r4" kind="internal">
  <rName name="information" synsetID="106250971" />
  <descr>the record stored in the system</descr>
 </Role>
 <Role id="r5" kind="internal">
  <rName name="register" synsetID="104883282" />
  <descr>stores borrower data</descr>
 </Role>
  <Port fromRole="r1" toRole="r2" card="1" />
  <Port fromRole="r2" toRole="r1" card="2" />
  <Port fromRole="r2" toRole="r3" card="1" />
  <Port fromRole="r3" toRole="r2" card="1" />
  <Port fromRole="r3" toRole="r5" card="1" />
  <Port fromRole="r5" toRole="r4" card="2" />
 </RoleModel>
 </OOramModelColl><?xml version="1.0" ?>

<!DOCTYPE OOramModelColl SYSTEM "OOramModelColl.dtd">
<!--  Created by XMLBuilder -->
 <OOramModelColl>
 <RoleModel id="m1" kind="roleModel">
  <Name>Wholesaler_Sub</Name>
  <descr>registers information about customer</descr>
 </Role>
 <Role id="r1" kind="stimulus">
  <rName name="customer" />
  <descr>buys parts</descr>
 </Role>
 <Role id="r2" kind="internal">
  <rName name="wholesaler" />
  <descr>has parts in store</descr>
 </Role>
 <Role id="r3" kind="internal">
  <rName name="shop assistant" />
  <descr>helps the customer</descr>
```

```
</Role>
<Role id="r4" kind="internal">
 <rName name="register" synsetID="104883282" />
 <descr>register of the customers</descr>
</Role>
<Role id="r5" kind="internal">
 <rName name="information" synsetID="106250971" />
 <descr>information about the customer</descr>
</Role>
 <Port fromRole="r1" toRole="r2" card="1" />
 <Port fromRole="r2" toRole="r1" card="2" />
 <Port fromRole="r2" toRole="r3" card="1" />
 <Port fromRole="r3" toRole="r4" card="1" />
 <Port fromRole="r3" toRole="r2" card="1" />
 <Port fromRole="r4" toRole="r5" card="2" />
 </RoleModel>
 </OOramModelColl>
```

Appendix E

Prolog Version of WordNet

In this appendix parts of the files in the Prolog version of WordNet are given to show how they are structured. I have used three files, and they contain the word senses, the relation between hypernyms and hyponyms, and the glosses. All Prolog predicates have a name and a set of arguments on the form: `predicateName(arg1, arg2, ...).`

E.1 Senses

Each line in the senses file wn_s.pl contain one predicate `s`. The following is an exerpt from the beginning of this file. The content of the parenthesis is in sequence:

1. a synset_Id—unique for each *synset*
2. the word's sequential position in a synonym set
3. the word form itself
4. the category of word (n—noun, v—verb, etc.)
5. the sequence in a set of meanings of a particular word form
6. a tag_state indicating whether the sequential position above is made due to frequency of use of the particular word form (1) or whether it is arbitrary (0). This attribute is not used by us.

```
s(100001740,1,'entity',n,1,1).
s(100001740,2,'something',n,1,0).
s(100002086,1,'life_form',n,1,0).
s(100002086,2,'organism',n,1,1).
s(100002086,3,'being',n,2,1).
s(100002086,4,'living_thing',n,1,1).
s(100002880,1,'life',n,10,1).
s(100003011,1,'biont',n,1,0).
s(100003095,1,'cell',n,2,1).
s(100003731,1,'causal_agent',n,1,0).
s(100003731,2,'cause',n,4,1).
s(100003731,3,'causal_agency',n,1,0).
s(100004123,1,'person',n,1,1).
s(100004123,2,'individual',n,1,1).
s(100004123,3,'someone',n,1,1).
s(100004123,4,'somebody',n,1,0).
s(100004123,5,'mortal',n,1,1).
```

```
s(100004123,6,'human',n,1,1).
s(100004123,7,'soul',n,2,1).
s(100008019,1,'animal',n,1,1).
s(100008019,2,'animate_being',n,1,0).
s(100008019,3,'beast',n,1,1).
s(100008019,4,'brute',n,2,0).
s(100008019,5,'creature',n,1,1).
```

E.2 Hypernyms

Each line in the hypernym file wn_hyp.pl contains one predicate `hyp`. The predicate has two arguments representing synsetIDs where the first is the spesialization (or hyponym) and the second is the generalization (or hypernym). The set of `hyp`- predicates ties all the synsets together in a hierarchical way.

```
hyp(100002086,100001740).
hyp(100002880,100002086).
hyp(100003011,100002086).
hyp(100003095,100001740).
hyp(100003731,100001740).
hyp(100004123,100002086).
hyp(100004123,100003731).
hyp(100008019,100002086).
```

E.3 Glosses

Each line in the file wn_g.pl contains one predicate `p` with the gloss for one synset. The arguments represent the synset's synsetID, the gloss, and sometimes also examples. Not all synsets have glosses. Note that each predicate below must be kept on one line, but for editing reasons, some of the predicates have been split over several lines.

```
g(100001740,'(anything having existence (living or nonliving))').
g(100002086,'(any living entity)').
g(100002880,'(living things collectively; "the oceans are teeming with
              life")').
g(100003011,'(a discrete unit of living matter)').
g(100003095,'(the basic structural and functional unit of all
              organisms; cells may exist as independent units of life
              (as in monads) or may form colonies or tissues as in
              higher plants and animals)').
g(100003731,'(any entity that causes events to happen)').
g(100004123,'(a human being; "there was too much for one person to
              do")').
g(100008019,'(a living organism characterized by voluntary movement)').
g(100008864,'(a living organism lacking the power of locomotion)').
g(100009457,'(a physical (tangible and visible) entity; "it was full of
              rackets, balls and other objects")').
g(100010123,'(an object occurring naturally; not made by man)').
```

Appendix F

Role Model Overview

In the tables below all the role models used in the experiments are listed. Table F.1 lists the role models used as targets in the experiments, while table F.2 contains all remaining models. The target models are sorted according to size and complexity. The names of the role models reflect the application domain, but they do not reflect the concrete task that is modelled.

		Index Structures		
Role model name	**Roles**	**1st**	**2nd**	**3rd**
LibrarySystem1	3	star	chain	chain
LibrarySystem1a	4	star	tree	chain
CustomerPlaceOrder	4	star	chain	chain
Program committee job no 2	4	star	chain	chain
Register for course	4	star	chain	chain
Request_Library_Card	4	tree	chain	chain
PaperSubmission	4	star	chain	chain
SeminarOrganizerSchema	4	tree	chain	chain
Library_sub	5	chain		
Library_sub1	5	chain		
Wholesaler_Sub	5	chain		
Checkbook application	5	star	tree	chain
Wholesaler_Sub1	5	star	chain	tree
Course Organization	5	tree	ring	chain
ConferenceOrganizationSchema	5	tree	chain	chain
CourseOrganizerSchema	6	tree	chain	chain
access_system1	8	star	tree	tree
Library1_a	8	tree	chain	chain
Library_a	8	tree	chain	chain
access_system2	9	ring	tree	tree
Library	9	chain	chain	chain
Library1	9	chain	chain	chain
Wholesaler1	11	tree	chain	chain
Wholesaler	12	chain	star	chain

Table F.1: Overview of the role models used as targets in the experiments

The tables include information related to the number of roles in each model and the structure indexes that have been stored in the repository. In the cases where the role model does not have three different substructures, the remaining locations are left empty. The 17 simplest role models have only 1 structure index.

Table F.2: Overview of role models used in the experiments

		Index Structures		
Role model name	**Roles**	**1st**	**2nd**	**3rd**
a system	3	star	chain	chain
access_system3	9	ring	tree	tree
Aircraft information	4	star	chain	chain
Airports at cities	2	chain		
AllocateResources	4	star	chain	chain
Apply for parking permit	4	tree	chain	chain
Appoint referee	2	chain		
ATM	5	ring	chain	tree
ATM class sys	8	tree	chain	chain
ATM perform transaction	7	tree	tree	tree
ATM scenario	4	chain		
ATM system no 1	7	tree	chain	star
ATM system no 2	7	star	tree	chain
Block connectivity object mode	6	tree	chain	chain
Booking system	4	chain		
Booking system1	4	chain		
Bureaucracy Pattern	3	chain		
car loan system	5	star	tree	chain
Car rental	2	chain		
car transmission	7	tree	tree	tree
ChangeARequirement	3	star	chain	chain
class consortium	3	star	chain	chain
Company with employees	3	star	chain	chain
Concert management application	4	star	chain	chain
Concert management application 2	4	chain		
Course test	3	star	chain	chain
Course test 2	3	star	chain	chain
Customer purchase and postal data	8	chain	tree	tree
decision maker sys	4	star	chain	chain
DefineProject	3	star	chain	chain
DefineProjectPhases	3	chain		
Derived traveling sys	6	star	tree	tree
desktop publishing system no 1	4	star	chain	chain
desktop publishing system no 2	7	star	tree	chain
Desktop publishing system no 3	6	star	chain	chain
Directed acyclic graph	4	star	chain	chain
Directed graph	4	tree	chain	chain
Directed graph 2	4	chain		
display gauge process sys	8	star	tree	tree

continued on next page

Table F.2: Overview of role models used in the experiments

		Index Structures		
Role model name	**Roles**	**1st**	**2nd**	**3rd**
employee register system	7	star	tree	chain
Employees and sports	2	chain		
enterprise project groups	6	tree	chain	ring
Flag nicknames	4	star	chain	chain
Flexible Manufacturing System	10	tree	tree	tree
Flight	7	tree	tree	star
Flight reservation model	11	tree	star	tree
Frontloading schedule	4	tree	chain	chain
Frontloading schedule 2	4	star	chain	chain
Glider simulator	10	ring	tree	tree
head office system	6	tree	tree	tree
Helicopter transport system	8	tree	star	tree
IFIP example - no 1	7	star	chain	chain
Inventory role model	7	tree	chain	chain
Library system	9	star	tree	chain
Library_sub0	4	chain		
Library_sub1	5	chain		
LibraryFull	5	star	chain	tree
LibraryFull0	4	chain		
Mail handling system	5	tree	chain	chain
ModifyStaffAllocation	4	star	tree	tree
Moon	3	star	chain	chain
Movie_Theater_1	5	chain	chain	chain
net connectivity object mode	7	star	chain	chain
node connectivity object mode	11	star	chain	tree
Open bank account	3	star	chain	chain
Organizing committee job 1	4	star	chain	chain
Organizing committee job 2	6	star	tree	chain
Organizing committee job 3	6	star	ring	chain
organizing committee job no 4	5	star	tree	tree
Organizing conference: Print attended badge	6	tree	chain	chain
Outboard sales store	10	tree	chain	chain
Paper Evaluation	3	tree	chain	chain
Pattern: controller	5	star	chain	chain
Pattern: tool	3	star	chain	chain
Pattern: view	5	tree	chain	chain
phone call	3	star	chain	chain
Program committee job 1: CFP	4	ring	chain	chain
Program committee job 3	7	star	chain	chain
Program committee job 4	5	star	chain	tree
Program committee job 5	7	ring	chain	chain
Program committee job 6	7	star	ring	chain
Program committee job 7	6	star	tree	chain
Program committee job 8	7	star	ring	chain

continued on next page

Table F.2: Overview of role models used in the experiments

		Index Structures		
Role model name	**Roles**	**1st**	**2nd**	**3rd**
Purchase system 2	4	star	ring	ring
Purchasing system	4	star	ring	chain
RegisterPaper	7	star	tree	tree
ROSA CASE Tool application	12	ring	tree	tree
ROSA CASE TOOL application no. 2	8	ring	tree	tree
scoring system	12	star	tree	chain
Simple diagram editor	7	star	tree	tree
Submit Referee Report	4	star	chain	chain
test no 3	3	star	chain	chain
Trader mechanism system	5	ring	tree	chain
Transaction counter	5	chain	chain	chain
Travel agent system 1	5	tree	chain	chain
Travel agent system 2	5	ring	tree	chain
travel permission sys	5	tree	chain	chain
University1	5	tree	chain	chain
University2	7	star	tree	chain
Updating account	3	star	ring	chain
user selections of elements	4	ring	chain	chain
Video Rental	9	tree	chain	chain
Video Rental Chain	9	tree	tree	tree
Video system	8	chain	tree	chain
View_and_control system	4	tree	tree	tree
Visual part architecture	12	chain	tree	tree
Wholesaler_Sub1b	4	chain		
writing paper	3	star	chain	chain

Errata

Page v “UML sequence diagrams” – Corrected to “UML communication diagrams” (in Abstract)

Page 5 “*Unified Modelling Language* (UML) sequence diagrams” – Corrected to ”*Modelling Language* (UML 2.0) communication diagrams (called collaboration diagrams in UML 1.x)”

Page 6 “diagrams like sequence diagrams” – Corrected to “diagrams”.

Page 6 “The UML sequence diagrams does not” – Corrected to ”The UML communication diagrams does not”.

Page 166 “UML 2 sequence diagrams” – Corrected to “UML 2.0 communication diagrams”.

Page 174 “collaboration diagram” – Corrected to “communication diagram”.

Page 175 “UML sequence models” – Corrected to ”UML communication diagrams”.

Page 175 “while the collaboration diagrams” – Corrected to “while the communication diagrams”.

Page 190 “UML *sequence diagrams*” – Corrected to “UML communication diagrams”.