NTNU
Norwegian University of
Science and Technology

Explanation-aware Case-based Reasoning

Marvin Bredal Lillehaug

Master of Science in Computer Science
Submission date: June 2011
Supervisor: Anders Kofod-Petersen, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Problem description

When tasks traditionally performed by humans are automated, it is important that the machines taking over are able to communicate how these tasks are solved and why. When an user is surprised by the time or the manner in which a task is executed, the system must be able to explain why the task was carried out in this way and at this particular time.

This project aims at investigating how intelligent systems, and case-based reasoning systems in particular, can become explanation-aware. Our aim is primarily to investigate existing case-based reasoning systems to see if explanation-awareness is achievable. Secondary, our aim is to develop a simple case-based reasoning engine that complies with our theoretical work on explanation-awareness.

Assignment given: 17. january 2011
Supervisor: Anders Kofod-Petersen

Abstract

As an increasing number of tasks are delegated to computerized and automated systems it is increasingly important that these systems are able to communicate exacly what operations are being performed and for what reasons. In this report we present a summary of the previous work done in regards to making systems explanation-aware, focusing on case-based reasoning systems in particular. Based on the background material presented we have implemented a plugin for Protégé 4.1 that generates explanations for the similarity values found in the retrieval step of the case-based reasoning cycle. In addition to this, the presented system is able to explain concepts used in the domain model by consulting external knowledge sources.

Preface

This report is the result of work done in the courses *TDT4500 - Specialization project in intelligent systems* and *TDT4900 - Masters degree in computer science* with a specialization in intelligent systems.

I would like to thank professor Anders Kofod-Petersen for his advice and input during the last year and Håvar Aambø Fosstveit for proof-reading this report.

Marvin B. Lillehaug
Trondheim, June 12, 2011

Contents

List of Figures

Chapter 1

Introduction and Overview

1.1 Background and Motivation

An increasing number of tasks are being automated and controlled by computers. This means that the users have less control over which operations are done and why, compared to earlier when the same operations were performed manually. This can is some cases lead to confusion since the user does not know the motivation behind the actions performed by the system. If the system performs actions that the user does not expect, confidence in the system's competence will decrease unless it can be justified that its actions are optimal. This can be done by putting effort into making the system *explanation-aware*, i.e. it has to be capable of explaining which pieces of knowledge has been used when a decision was made, and why it was correct to use this piece of knowledge in this particular way.

We are in this project focusing on explanations in the context of `case-based reasoning (CBR) systems`, since this type of system usually represent its internal knowledge in a way that makes it possible to reason with and thus generate explanations. The background material presented in this report focuses on the CBR aspect of explanations, but is valid for all types of systems where it perform some action that is hidden from the end user.

When doing research for the background information we also discovered a link between the knowledge representation and explanations to `the semantic web` and have chosen to use this as an angle for our project.

1.2 Goals and Research Questions

Goal 1 Find existing solutions and methods for explanation-aware and mixed-initiative systems in the CBR and expert system literature.

In order to be able to fulfill this goal we have to acquire knowledge about which systems have been successful and what methods where utilized in these systems. This means that we have to find literature describing the following topics:

- Explanations in expert and CBR systems
- Mixed-initiative in expert and CBR systems
- How to represent knowledge in order to be able to reason with it
- CBR systems in general

In the case of explanations and mixed-initiative we have to be able to account for both their general role in expert systems, and the concrete solutions for generating them.

Goal 2 Design and build a CBR system that is explanation-aware and use OWL as knowledge representation. I.e. Both the case base and knowledge is contained in the ontology.

The motivation for focusing on OWL will become clear when this technology is presented in chapters 2.6.9 and 2.6.10.

Research question 1 Are there any benefits from using OWL over Frames to represent ontologies?

We do not have ambitions of contributing greatly to this field of research, merely understand what work has been done earlier and summarize this.

1.3 Research Method

Since our knowledge in this particular field is limited, we will have to do a survey of the work previously done and which systems exist. The literature relevant to us reviews the basic concepts and workings of *case-based reasoning*, as well as the work done regarding how to make the resulting system explanation-aware. The main keywords used when searching, in combination with *case-based reasoning* are shown in Table 1.2, categorized by the sub-domain. The keywords where selected based on the problem description and the fact that in order for the system to be able to reason with its contained knowledge it has to be represented in a suitable manner.

In addition to this the curriculum for in the courses `TDT55` and `TDT70` (2010) where used. The curriculum in the mentioned courses are listed in Appendix .1

and Appendix .2 respectively. Many of these articles where also among the results when performing the searches.

When an article was considered relevant, the articles known to cite the article were also included in the background material if they where considered relevant, e.g. further work and improvements. If a concept introduced in an article was unfamiliar to us, we followed citations "backward" in order to get a more fundamental description of concepts and methods.

In this context an article is `relevant` when one of the following criteria is fulfilled.

- Provides background information that we needed in order to understand the material presented or felt should be included in the background chapter the give the reader the necessary background information.

- Describes progress that gives a better understanding of the covered field or in other ways present results that are relevant in the context of explanation-aware CBR.

An article was included when, after reading the abstract, it seemed likely that it contributes to the background of the field and provided further description of the concepts that are important for the subjects discussed in this report. Otherwise it was rejected and not included as background material. Articles we considered likely to contribute typically used words such as explanation-aware; roles of explanation; ontology; and other words that where descriptive for the topic of interest and similar to the keywords listed in Table 1.2.

When performing the literature search we have used *Google Scholar*[1]. This is a search engine that has indexed a large number of scientific digital libraries and search engines. If Google Scholar had not existed we would have performed the same searches in the digital libraries listed in Table 1.1. Most of the relevant results did in fact originate in these libraries. The search string used was *("case-based reasoning" or "cbr") and "<keyword>"* where `<keyword>` corresponds to a term from Table 1.2. The number of results for the each of the resulting search strings are show as a number behind each term. Since the results are ordered by decreasing relevance only the articles on the ten first pages of the search was considered.

When finding information on OWL we primarily consulted the specification and the resources listed in the Protege wiki. [2] To obtain resources about merging/mapping/matching the query `(mapping OR merging OR matching) ontology survey` was performed in Google Scholar.

[1] `http://scholar.google.com`

[2] http://protege.stanford.edu/doc/users.html

Source	Url
ACM Digital Library	`http://portal.acm.org/dl.cfm`
IEEE Xplore	`http://ieeexplore.ieee.org/Xplore/guesthome.jsp`
Springer Link	`http://springerlink.com`
ScienceDirect	`http://sciencedirect.com`
Wiley Inter Science	`http://interscience.wiley.com`
CiteSeerX	`http://citeseerx.ist.psu.edu`
ISI Web of Knowledge	`http://isiknowledge.com`

Table 1.1: Sources used when searching for background information

Explanation	Knowledge representation	Mixed-initiative
Explanation (~4510)	Knowledge representation (~3580)	Conversational (~1010)
Explanation-aware (~22)	Knowledge-intensive (~1480)	Mixed-initiative (~416)
	Knowledge-light (~122)	

Table 1.2: Keywords used in search

1.4 Report Structure

We have just presented the method by which we have researched the field. Next we will present the concepts we consider important to understand the field. After we have understood the concepts and principles that are important for explanation-aware systems and case-based reasoning, we will present the system we have implemented to test the concepts described in the background material. When this is done we present an evaluation of the implemented system and finally our conclusions and further work.

Chapter 2

Theory and Background

Since the inception of the discipline of computer science in the 1950s and early 1960s [Denning, 2003], a lot of effort has been put into the subject of enabling the computer to utilize all the stored information in solving a novel problem. Many different approaches have been tried and some of them have failed because their performance has not been good enough. In addition to the topic of effectively making use of the information contained in a system such that answers are generated in a timely fashion, a lot of resources has also been spent researching how the systems can explain what they are doing and why. The explaining aspect of expert systems, as well as any other kind of computer system, have been a research topic since these kind of systems has been considered successful. [Shortliffe et al., 1975, Tintarev and Masthoff, 2007]

In this section we will look at some of the methods that have had some success and what their limits are. The main focus will be on so called *Case-based reasoning (CBR) systems*. But before we get to these systems we will describe some other approaches. We will also get into the role of explanations in relation to these kind of systems, why they are important and what challenges are associated with creating them. We go on to describe the general role of explanations. Then we describe the knowledge systems, first the traditional and then CBR systems. We then explain how explanations are given in the different systems. After this, we will get into a special kind of CBR system that tries to involve the user by conversation. Lastly we present theory regarding knowledge representations, which is concluded by focusing on the knowledge representation used in *the semantic web*, OWL.

2.1 Explanations

2.1.1 Why bother to explain?

In order to be able to use a piece of information, one has to know how it relates to other things. There is a significant difference in answering the question "what does the grep-program do?"(search for patterns in a text) and being able to use the program effectively. In order to answer the above question, all that is needed to know it that it is searches through files and print lines that matches a given pattern. But in order to use it properly, we need to know about regular expressions and how to specify the parameters correctly. This can be seen as an analogy to information with and without explanations. In order to move from the level where one knows what a concept is to the level where it is possible to use this concept in combination with other things, utilizing the true power of the concept, one has to know how it fits in the context of other concepts. In order to manage this, explanations has to be given as support to the answers generated by the system.

In addition to this, the users of the system are more likely to trust it when the system can explain why it has reached a particular answer[Ye and Johnson, 1995]. Not giving an explanation is some times better than explaining the answer. If the system is not able to generate an answer that the user is able to understand it is likely that it will contribute to make the user more confused than he was prior to getting the explanation. Another example is that the explanation is simply stating the obvious. "A cake is a form of pastry" is not very likely to be a useful explanation since this is common knowledge (unless the user has a poor knowledge of the English language). In this case it was perhaps not better to leave the explanation out, but nothing was gained by giving it. [Cunningham et al., 2003]

2.1.2 Goals and kinds of explanations

As mentioned, there is no point in giving an explanation if the contents of it is not understandable to the user, regardless of whether it is valid and can be considered a good explanation. The same is the case if the content of the explanation is not relevant to what the user wants or needs to know. When investigating a robbery, the detective does not care that the person committing it used to have a dog called Alfred. Only the person's identity and perhaps his location is of interest. The detective does not care why the robbery was performed, since all robberies are done with the intention of getting more money or goods. From these examples it should be clear that the quality of an explanation depends on what kind it is, and what the goal of the explanation is. This is what we will try to describe next.

When constructing an explanation we have to consider who the receiver of the explanation is and at what level of understanding he is. It is of no use trying to

explain using advanced concepts if the meaning of these are unknown to the receiver. Likewise it is unnecessary to explain something that is already understood. This aspect of explanation, user modeling, has received a lot of attention[Wahlster, 1988, Kobsa, 1993] but will not be discussed in detail here. We will assume that there is some way to determine at what level explanations should be given.

Spieker [1992] proposes five intuitive types of explanation that are useful in the context of explanation systems. We will now summarize these.

Conceptual explanations explain the meaning of a concept, and how it is used. Explanations of this kind often try to connect unknown and known parts of knowledge, explaining the unknown in terms of the known.

Why-explanations as the name suggests, explain why something has occurred or why something is the way it is. It can also be seen as a way of getting further down in the causal and conceptual structure. Always demanding a why-explanation will make the system try to explain until it is at the edge of its knowledge.

How-explanations explain the causal chain that lead to an event. These can be seen as a special case of why-explanations, explaining in what order events happened rather than why.

Purpose-explanations describe the function of an event or item. E.g. "a pencil is used to write".

Cognitive explanations explain or predict the behavior of 'intelligent systems' on the basis of known goals, beliefs, constraints, and rationality assumptions. If it is not clear what kind of explanation the user expects it is quite hard to generate a satisfactory explanation. If a conceptual explanation is needed, the user will probably not understand the content of an explanation of some other kind, since they may be using concepts that has not been introduced to the user.

Another aspect of explanations is what Sørmo et al. [2005] calls explanation goals. The goals and kinds overlap in some areas, but for the most part they highlight different aspects of explanations.

Transparency show how the system reached the answer. The easiest way to fulfill this goal is to simply print the trace of the reasoning the system has done, and which rules where used, from the point where it got the query to the answer was show to the user. In most cases this simplistic approach will only be useful for an expert trying to validate the reasoning of the system. A normal user will

not have the required domain knowledge or understanding to comprehend what the trace means.

Justification is needed in order to make the user more confident that the system comes up with reasonable answers, and that these answers are valid. For instance explaining why one rule was used instead of another, or why two cases are similar. This may in some cases be quite similar to transparency, in each decision point displaying each candidate and output the attributes used to decide which candidate to use. In many cases what is shown to the user is two or more candidates and a number representing some attribute to base the decision on, most likely similarity, ranging from 0 to 1. For this type of justification the user has to manually inspect the two candidates in order to understand what makes them different and why one was chosen over the other.

Relevance is very similar to the "why-kind" of explanation. Roth-Berghofer and Cassens [2005] describes the terms for explaining why a question asked by the system is relevant to the context. It can also be in relation with justification, explaining why comparison of a particular attribute is done in a certain manner.

Conceptualization means the same when defined as a goal as it does when defined as a kind of explanation. It is the process of forming concepts and the relation between concepts.

Learning is defined as an explicit goal by Sørmo et al. [2005], meaning that the system should teach the user about the domain at hand. Since all the other goals in some way involve learning, this seems a bit redundant. The way this goal is described it seems as if it only applies in tutoring systems, not only finding a good solution but also explaining the solution to the user in a way that gives the user a better understanding of the domain. So it may be that this goal is simply describing a explaining system that generates comprehensive explanations by mixing transparency, conceptualization, and justification.

In addition to the mentioned goals and kinds, an explanation must fulfill the following requirements, put forth by Swartout and Moore [1993].

Fidelity means that a constructed explanation should mirror the knowledge that has been used by the system when reasoning, as well as when the explanation was constructed. This is related to the transparency goal, an explanation with high transparency has high fidelity.

Verification is needed in order for the knowledge engineer to know that the system works in the way intended. I.e. it must be possible to verify that the system performs its tasks correctly, does not use resources unnecessarily, and simply work as it is supposed to.

Duplication is quite similar to the learning goal. The system should expose its knowledge and make it possible for the user to understand the methods and knowledge used when reasoning and constructing explanations.

Ratification means that the system should behave such that the user becomes more confident in the results presented. This is related to the justification goal and is crucial when making a system that is supposed to be used in a production setting, since the system is used in a manner where the user does have knowledge about the inner workings of the system. Thus the only thing determining the confidence in the system is what it presents and how it is presented.

Low construction overhead is important. There will always be a finite amount of resources available for generating results. The user will not accept having to wait a long time for the results, and the amount of computing power will always be limited. However, as the amount of knowledge in the system increases, the overhead also increases due to the fact that the system needs to go through more and more knowledge in order to find what is relevant. This is a problem that has been given much attention in the field of expert systems.

A similar set of goals for recommender systems

In [Tintarev and Masthoff, 2007] the authors present a set of goals that are quite similar and partially overlapping the ones just mentioned, but aimed at evaluating the explanation facilities in some selected recommender systems.

Transparency is presented in the same way as we already have presented it, how the given answer was reached.

Scrutability involves being able to tell the system that it is wrong, especially important if the system collects and interprets information on its own initiative and in the background. This involves both correcting the answers that are given and correcting the assumptions made when creating these answers, should they be unreasonable assumptions.

Trust in the system is achieved when the system either gives good answers or manages to justify why the quality of the answer was as bad as it was. This goal can be seen as an equivalent to the `justification` goal mentioned earlier. The authors does however also refer to a study that shows that a large portion of the credibility of a system (a web page in the referred study) was due to the appearance of the system. This suggests that a system may be easier to trust if its design and user interface is intuitive.

Effectiveness involves making explanations that, instead of convincing the user to choose an item, assists him such that he makes better decisions. This can be seen as equivalent to the `learning` goal defined by [Sørmo et al., 2005].

Persuasiveness is the ability to convince the user to "accept" the suggestion. This is not quite the same as justifying the answer, but more about manipulating the user such that the recommended items appear more correct than they in some cases may be.

Efficiency involves enabling the user to make decisions faster than he would had he not been given an explanation. For instance by explaining how two competing items differs.

Satisfaction naturally describes how satisfied the user is with the experience and the usefulness of the system. One could argue that this does not deserve to be a separate goal, but rather something that simply should be mentioned when describing the `trust` goal. In any case this is an obvious goal for any system, and systems designer that does not wish to maximize the satisfaction of the user is likely to not be rational.

2.1.3 Evaluation of explanations

When evaluating whether an explanation is good or not, we clearly need to have the above requirements and goals in mind. When an explanation does not address any of the goals, or fulfill the requirements put forth, it is most likely an insufficient explanation. Leake [1991] developed the following nine dimensions by which explanations can be evaluated.

Predictive power The fact that an event is caused by some event, does not necessarily mean that the cause is predictive by that event. A particular tire-blowout may have happened when the vehicle was driving at high speed, but the tire did not blow just because of the high speed. The high speed was just a trigger,

causing the tire to blow at this particular moment. So we can say that high speed has low predictive power in regards to tire blowouts. The explanation that the tire was very old and the vehicle was driving at a high speed on the other hand, has significantly higher predictive power.

Timeliness In order for us to be able to act on the information in an explanation, it needs to explain something that happens a sufficient amount of time before the explained phenomenon. "The window broke because it was hit by a stone" is not an explanation that will make us able to react to the same event if it occurs again in the future. "The window broke because it was hit by a stone thrown up by a passing car" however, tells us that when we are driving and see a oncoming car (and there are stones in the road), we should pull a bit to the side to reduce the chance of getting hit by stones.

Knowability The explanation that a tire blowout happens when driving at high speed with old tires are not doing us any good if we for some reason are unable to know whether the tires are old or not (hopefully we are capable of knowing whether we are driving fast or not). I.e. no matter how early a predictive event may be, it will not help us to predict unless we can find out that it occurred. The event is in this case that the tire has become old and worn.

Distinctiveness In order for a cause to have predictive power for a surprising event, the cause itself must be surprising. In other words, the cause has to be unusual compared to the expected situation, so that it gives evidence for the surprising event as opposed to the previously expected outcome.

Causal force The explanation using only high speed to account for the tire blowout has very little causal power. This is because the portion of high speed trips that end in a blowout is very small, and that the portion of tire blowouts that happen in high speed is small as well. I.e. The antecedent of the event must have caused the event to happen. The connection between events have to be causal, like when heart failure causes death. When the connection between events is not that strong it may be predicting. The fact that old cars often have worn tires only predict that the tires are worn, we do not know the reason this happens (being naive and not knowing anything about cars and tires).

Repairability Just as an explanation does not help us prevent future surprising events if it cannot predict it, it does not help us if we cannot do anything to stop it even though we know it is going to happen. Neither of the explanations mentioned above will help a person that has been kidnapped and been placed in the trunk.

The person can not prevent the car from driving or do anything to mend the old tire. In any case there would not be any point trying to repair the tire, it would probably be better to replace it entirely.

Independence There is no point in fixing a problem if something will cause the problem to reappear as soon as we fix it. When a fuse is blown due to a short circuit, there is no point in replacing the fuse since it will be blown as soon as we turn on the power. This illustrates that the explanation needs to trace back to a cause with independence from prior causes.

Blockability In the example of the kidnapped person, the person in the trunk had no way of stopping the blowout. The kidnappers on the other hand could have blocked it by not using the car (or not kidnap the person in the first place).

Desirability By identifying actors' contributions to an event and ascribing praise or blame for their roles, it may be possible to influence their future behavior. This can be done according to the desirability of the outcome, and of actions leading to it. We may blame the blowout on the driver of the vehicle since it is the drivers responsibility to check that it is in a condition where it is safe to drive.

2.2 Explanations in expert systems

As mentioned in Section 2.1 there are several kinds of explanations a system can generate that help different users to different degrees. As well as having to consider the kind of explanation to construct, the quality of the explanation has to be adequate and it has to satisfy the goals the user has for the explanation. For the reasons mentioned earlier much research has been done on the subject of making systems explain what they are doing and why. As the systems grow more and more complex the need for explanations also grow since more tasks become controlled by computers. In order for the users to understand why the butler-bot serves soda instead of beer, and to know what needs to be corrected, we need to know what kind of knowledge was used to make the decision and what reasoning steps it went through.

The transparency goal is easy to fulfill for any system and when a user has only this goal in mind, it may be reasonable to think that he wants to verify that the system works as it should. What is needed to fulfill this is some way to monitor and trace the execution of the system, i.e. which methods are used and what knowledge is used during the execution. The output from this is in most cases not

very understandable for a novice user, depending on the formating. One example is given in Figure 2.1 where output from an execution of the Inference Web system [McGuinness and Silva, 2003] is shown. The example shown uses rules "drinks used for seafood meals are restricted to white drinks" and "the new meal is seafood" to show that only white drinks should be used for the new meal. For a user that knows first order logic, the reasoning trace shown may be understandable, but a novice user will probably not understand what is going on. This clearly shows that for an explanation constructed by a system to be useful, in other ways than to verify the behaviour, some other type of explanation is required. This does however not

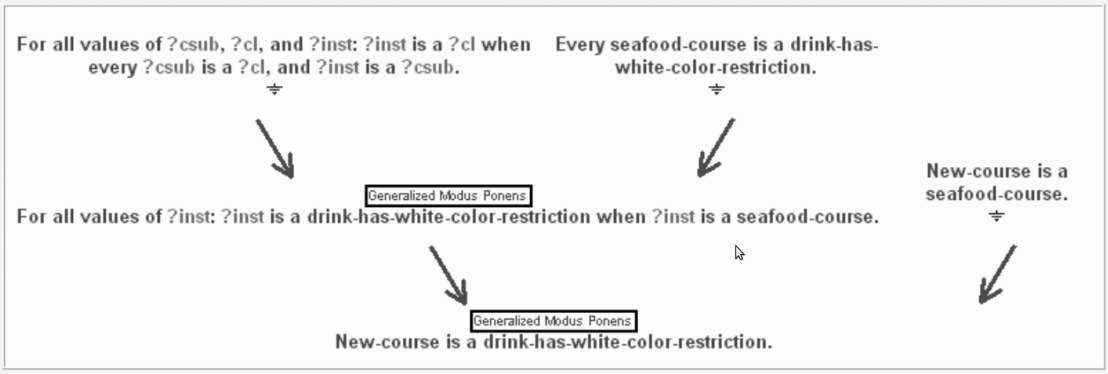


Figure 2.1: Output from a run of the *Inference Web* system [McGuinness and Silva, 2003]

mean that transparent explanations are completely incomprehensible, as MYCIN [Shortliffe et al., 1975] is a good example of. MYCIN was a interactive computer program for advising physicians regarding selection of antimicrobial therapy for hospital patients with bacterial infections. Knowledge in MYCIN is stored in rules that has a set of preconditions that, if true, results in an action or conclusion as shown in Figure 2.2.

MYCIN had a sub system for constructing natural language from these rules and was also able to understand simple queries written in natural text. This enabled MYCIN to engage in a dialog with the user where the user first was asked a series of questions that were used to determine what infection the patient was suffering from. When MYCIN had provided a conclusion, it was possible for the user to ask "how" and "why" in order to make the system explain why a particular question was asked (the purpose of the question) and what knowledge was used in reaching a conclusion respectively. Since MYCIN is an interactive system, asking and answering questions, we can say that it is a conversational system. We will come back to such systems in section 2.5. MYCIN has however quite limited capabilities to reason about things that are not explicitly stated in rules. This has been tried solved by, among other, NEOMYCIN. See [Clancey, 1983] for a survey

IF: 1) THE STAIN OF THE ORGANISM IS GRAMNEG, AND
2) THE MORPHOLOGY OF THE ORGANISM IS ROD, AND
3) THE AEROBICITY OF THE ORGANISM IS ANAEROBIC
THEN: THERE IS SUGGESTIVE EVIDENCE (.6) THAT THE IDENTITY OF THE ORGANISM IS BACTEROIDES

Figure 2.2: A rule used in MYCIN [Shortliffe et al., 1975]

and description of these. We refer to [Shortliffe et al., 1975] for details on MYCIN.

It is not exclusively in communication with the user explanations are helpful. When the system is reasoning it must often have the ability to explain to itself why something is the way it is. When conclusions can be justified by explanations grounded in domain knowledge, rather than some inductive bias they are more reliable since they are able to justify the generalizations that they make [Mitchell et al., 1986]. It is much easier to create justifications while doing the reasoning than at explanation-time backtracking in order to find justifications. When systems utilize explanation-based generalizations instead of similarity-based generalizations they are able to learn more information about a single concept from a single example than it would otherwise[Mitchell et al., 1986]. The concept of *explanation-based generalization*[1] is not very important in the context of CBR systems since we rarely try to generalize, it may however become relevant in regards to knowledge acquisition.

Mozina et al. present another method that augment the knowledge acquisition process. In *argument based machine learning* the expert provide support for a particular example by providing arguments for it, this increases the amount of knowledge in the system as well as the accuracy of the domain model. An argument can be of two types, positive and negative, depending on whether it argues that the example has a certain class or not. The system maintains a current hypothesis and when there are training examples the hypothesis cannot explain, it asks the expert for arguments that either changes the class of the example or support the current classification. The system also makes use of counter examples in order to point out where the argumentation for an example is insufficient. By iterating this behavior the authors claim the hypothesis continuously gets better and produce rules that are more intuitive and logical than those produced by methods using inference.

2.3 Case-based reasoning systems

Expert systems were among the first truly successful forms of AI software [Russell and Norvig, 2009, pp. 22-24] [McCorduck, 2004, pp. 327-335,434-435]. The

[1]Dejong and Mooney [1986] states that this is not the same as explanation-based learning, as well as point out shortcomings in Mitchell et al. [1986].

systems that have been developed are based on several different principles that all work well for certain things. We will in this paper focus on *case-based reasoning(CBR) systems*.

What separates CBR from other types of systems is that it maintains a library containing past cases and uses these as a basis for solving the problems that are presented to it. In this context a case is an example of a situation that has occurred in the past, together with its solution, that has been stored for future use. To clarify the concept we will now describe the process of solving a problem from the cooking domain.

You are having a small dinner party and are now in the process of deciding what to make for dinner. You do not have a cookbook, only your CBR system that has recipes in the case base. Since it is a small party you only want to make a main dish. After opening the interface to the CBR system you specify that the meal should be something Indian and with chicken. The system now starts to execute a four step cycle, consisting of the steps *retrieve*, *reuse*, *revise* and *retain*[Aamodt and Plaza, 1994]. In the retrieve-step the case base is searched in order to find

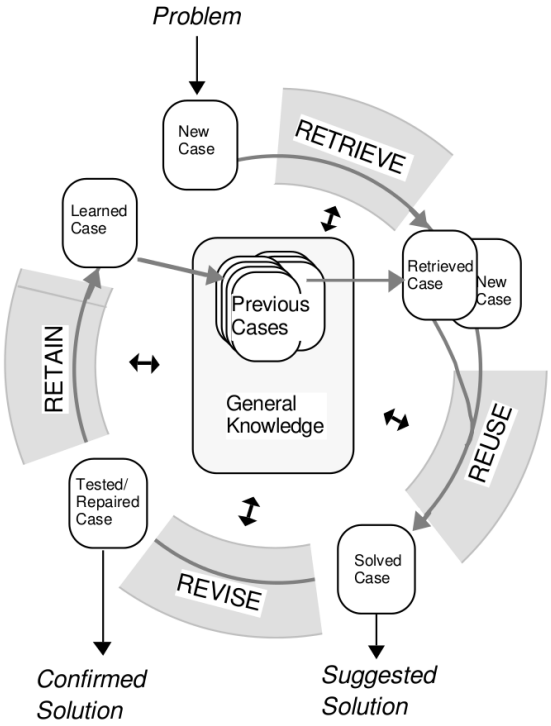


Figure 2.3: Illustration of the CBR cycle [Aamodt and Plaza, 1994]

an earlier case that is similar to the current one. The most prominent problems associated with this step is how to measure the similarity of two cases and how to structure the case base so that finding the appropriate cases remains possible

and efficient as the case base increases in size. Much effort has been put into solving these two problems and many approaches to similarity measures as well as indexing methods has been proposed. When the case base is large and it is not organized in a satisfactory way, this results in the swamping problem[Smyth and Keane, 1995, Jr and Ram, 1993]. The swamping problem occurs when the retrieval process have to use much of it time searching the case-base for potential matches, it contains so many cases that the retrieval result is not worth the time it took to retrieve it. We will come back to measures, in addition to indexing, to combat this later, in the retain-step.

In our example the system looks in the case index for cases indexed in "indian" and "chicken" and finds several matches. Our case is only partially specified having only two attributes, meat type; and country of origin. This will usually result in a large number of matched cases and the system will have a hard time determining what cases are best suitable. The cases that are matched get sent to the next step in the cycle. It varies from system to system whether only one or several cases are passed to the next step.

The next step in the CBR cycle is the reuse step, and as the name suggests the goal is to reuse the past cases in constructing the solution to the current case. How this is done varies from system to system and depends on several things, amongst others, how many cases are passed on from the retrieval and how complex the reuse step is. The simplest approach simply takes one case and use its solution directly as the solution to the current problem. Other, more complicated approaches may involve solutions from several cases being combined in order to construct the solution to the current case. In this case, several things has to be taken into account, like recognising the difference between the cases and how the different attributes related to each other. When the solution is finalized it is passed on to the next step, revision.

Our system has retained only one case from the case base. It could not find any prior cases that matched both chicken and India, but it found a case that matched India and that contains pork. Now, using available domain knowledge, the system reuses the retained solution and adapt it by changing pork to chicken since it knows that both are meats. The solution is then passed on to the next step in the cycle.

In the revise step the proposed solution is evaluated, either by the user or by some other part of the system. A common "in-system" evaluation is to simulate how the solution will work out in the real world. Unfortunately this is in most cases very hard to achieve. When it is the user's responsibility to evaluate the solution he has to input what the results of applying the found solution was. In some cases this may take a very long time.

The results of the evaluation can be either success or failure. If it was a failure

we can try to repair the solution, by applying domain-specific knowledge or with help from the user. After a repair we may evaluate the solution once more, or until we either succeed or give up. When we have found a successful solution we move on to the next step in the CBR cycle. Simulating the process of cooking and eating our Indian chicken dish in an appropriate way seems like a intractable problem, so we will have to evaluate the proposed solution manually. If we deem the meal a failure we could try to repair the recipe by for instance trying some other kind of meat, or identifying why it was a failure and try again. We could also give up and pass the failed case on to the next step. In the case where the solution was satisfactory we also pass it on to the next step.

The last step in the cycle is the retain step. This is the process of learning from the problem-solving attempt by finding out if it contains knowledge that is not already existing in the system. There are many approaches to retaining cases. Most systems analyze the new case in order to determine whether it should be retained or not. If the new case does not contribute a significant amount of new knowledge to the system there is no point in retaining it. In addition to this there are many challenges associated with retaining. For it to be possible to maintain short retrieval time it is important to have a good policy for which and how cases should be stored in the case base. If the cases are not indexed in a good way or if new cases bring no new knowledge into the system's knowledge, the retrieval step might be inefficient and even though the relevant knowledge is existing in the system, it may not be possible to find it[Smyth and Keane, 1995, Jr and Ram, 1993]. There are many attempts at solving this, but these are considered outside the scope of the paper and will not be discussed further.

2.4 Explanations in CBR

In traditional CBR the only explanation available is that which shows how the retrieved case relates to the problem case. This may fulfill the transparency goal, but it does not justify the case in other ways than the similarity metric. As mentioned earlier, it is harder to provide a good explanation if it has to be generated from scratch when it is needed. Schank and Leake [1989] describes a system that use *memory organization packets*[Schank, 1983] and *explanation patterns* to represent explanations and cases. The system works so that it requires an explanation when an unexpected event occurs, i.e. an anomaly occurs. In order to fulfill the required properties and explanation pattern must contain the following attributes.

- What the pattern explains.
- Under what conditions is the pattern likely to be valid.

- Under what conditions is the pattern likely to be useful.
- Relation among beliefs, showing why the anomaly might have been expected.
- A summary describing when the pattern can be used in planning.
- Prior episodes that have been explained by the pattern.

When a good enough explanation has been constructed it is generalized and stored in the case base. By generalizing it is possible to reuse the explanation in a broader context. The example given is that the death of a jogger may be used as basis for explaining the death of a race-horse, since the circumstances of both deaths are similar. As mentioned earlier, domain knowledge is needed in order to create the generalization. Using the generalized concepts underlying the anomaly as well as some of the key concepts involved in the cause, makes the index efficient and retrieval of prior anomalies possible.

2.4.1 CREEK

The role of domain knowledge has been mentioned a few times already, and it seems like in order to be able to satisfy more than the transparency goal we need domain knowledge. *CREEK* is a knowledge-intensive CBR where much of its explanation capabilities comes from the underlying representation of the knowledge. All knowledge, cases as well as concepts and relations, are represented by frames that are densely coupled in a semantic network[Aamodt, 1994]. When CREEK receives a new problem case it goes through a cycle consisting of three steps, *activate*, *explain* and *focus*. The first step looks at the input case and follows links in the semantic network either through spreading activation along relations or direct "reminding" (a link to a similar case) to past cases. The activated knowledge is then used to generate goal hypotheses, and explanations for these, that are justified by knowledge in the network. When creating explanations, CREEK first tries to explain away the differences in the retrieved and the input case, then it tries to explain the similarities. The last step takes all these hypothesis and based on the explanation gives a single solution suggestion. The explanations that are created during this cycle are nested structures of relationships. When a suggested solution is accepted, these explanations are retained as part of the solution. CREEK achieves three of the explanation goals. Transparency by visualizing how the retrieved case matches the problem case and how they are related. Justification by providing the possibility to compare the similarities and differences between the case, both syntactically and semantically. Since all knowledge is organized in a semantic net, the conceptualization goal comes for free [Kofod-Petersen, 2008].

2.4.2 myCBR

myCBR[2] is a case-based reasoning tool developed at The German Research Center for Artificial Intelligence (DFKI). It currently exists in two versions, 2.6.6 and 3.0-beta. The former is a plugin for Protégé 3 and includes means for executing queries, defining similarities, and the support for defining explanations for the concepts already defined in the system. The latter version is a standalone framework for constructing CBR-applications independent from Protégé, seemingly intended to build standalone applications.

myCBR 2.6.6 provides three kinds of explanations. `Concept` explanations, `forward` explanations and `backward` explanations. Conceptual explanations are used to explain the vocabulary of the system. These explanations are added manually to the system and saved in a separate file for explanations. It is possible to add urls, but as we understand it the system does not try to fetch explanations from these, but merely acts as a reference for the user. Backward explanations explain the outcome of a particular retrieval result and provide means for understanding the results of a similarity calculation. This explanation is built by constructing a tree that mirrors all similarity measurements performed during a retrieval. This contributes to achieving both the `transparency` and `justification` goals. Forward explanations are intended to assist during modeling and maintenance, by providing information about the status of the model. This is implemented such that, while the user is constructing a similarity measure, he is shown what the results of a retrieval will look like. [Roth-Berghofer and Bahls, 2008, Bahls and Roth-Berghofer, 2007]

The later version of myCBR does however not support explanations that well. It does have some source files supporting explanations, but not to the same degree as the previous version. This is something we intend to improve as part of our work, and will come back to in Chapter 3.3.

2.4.3 jCOLIBRI

The aspect of explaining the results in any of the steps in the CBR cycle has not been incorporated into jCOLIBRI's[3] architecture. There is nothing keeping track of what the similarities between a case and the query comprises. This means that an application built with jCOLIBRI as a foundation will have to make changes in the code in order to keep track of what has been done during the CBR-step in question. When one has the required programming skills this is however not a problem. Pedersen [2010] and Gravem [2010] presents an extension done to jCOLIBI such that it explains the retrieval results in terms of how the different

[2]http://mycbr-project.net

[3]http://gaia.fdi.ucm.es/projects/jcolibri/jcolibri2

attributes in query and case are similar and to what degree, as well as why the retrieved cases is a good answer.

2.4.4 Knowledge-light CBR

CREEK is, as mentioned, a knowledge-intensive system where the cases are "submerged" (exist in the same knowledge base) in domain knowledge. An example of a knowledge-light CBR system is Strategist [McSherry, 2001]. It organizes its cases in a decision tree-like structure and uses this structure, rather than domain knowledge, to generate explanations. The relevance of attributes are in this system described in what the authors call "strategic terms" [McSherry, 1999], which means that rather than a single split criterion like information gain, splitting is based on five splitting strategies. We refer to McSherry [1999] for the details.

In order to compare how transparency explanation compared to no explanation and a rule-based explanation, Cunningham et al. [2003] developed a knowledge-light system using WEKA [Hall et al., 2009]. Their conclusion was that case-based explanations are slightly better than rule-based explanations which in turn are marginally better than no explanations.

Similar research has been done by [Doyle et al., 2004]. They developed a knowledge-light system in order to investigate the notion that it is not necessarily the nearest neighboring case, but the case that lies between the query case and the decision boundary that is best suited as basis for an explanation. First they classify using nearest neighbors, then they use `explanation utility` to reorder the neighbours and presenting them. We refer to [Doyle et al., 2004] for the details of the `explanation utility metric`. This approach, in addition to the Knowledge-light explanation framework (KLEF) is presented in Nugent et al. [2008]. KLEF breaks the retrieval and explanation process into five phases. First, logistic regression is used in order to gain more insight into the relationship between input variables and a target, or class variable. A `local case base` is then created, containing cases from both sides of the decision boundary. The next phase finds a-fortiori explanation cases that lie between the query case and the decision boundary by using the logistic model to calculate the probability that each case is of a certain class. The final phase attempts to describe any feature-value differences that might exist between the query and the explanation case. It does this by using the logistic model to extract information about the influences of feature-values on the class value in terms of odds ratios. The results they obtain from the implemented system is that KLEF is able to produce explanations that increase the users confidence and the level of satisfaction compared to simple case-based explanations (i.e. simply presenting the most similar case).

2.4.5 Knowledge containers

Richter [1995] notes that CBR systems has their knowledge in four types of containers, vocabulary; similarity measures; adaptation knowledge; and case base, each providing their part to the explanatory capabilities of the system. In Table 2.1 we can see how Roth-Berghofer [2004] mapped the four knowledge types to the kinds of explanations mentioned earlier and we refer to this for a more detailed description of the mapping. Figure 2.4 illustrates how the containers relates to one another.

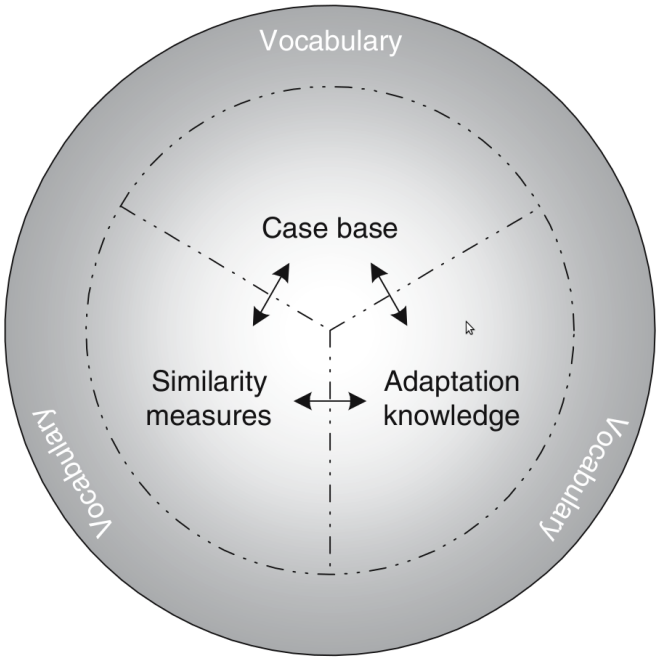


Figure 2.4: Knowledge containers in a CBR system Roth-Berghofer and Cassens [2005]

The vocabulary defines attributes, predicates, and the structure of the concepts that can be represented in the system. The organization and content of the vocabulary specifies how different concepts, cases, attributes, and properties the other containers are made up of, and thus forms the basis for these.

The similarity measures are crucial in a CBR system, since they dictate which cases are retrieved and used for further processing.

Adaptation knowledge is knowledge about how one case can be transformed into another. In essence it tells something about which attributes it makes sense to change in a case in order to make it a better suited solution.

Knowledge container	Contributes to
Vocabulary	conceptual explanations, why-explanations, how-explanations, and purpose explanations
Similarity measures	why-explanations, how-explanations, purpose explanations, and cognitive explanations
Adaptation knowledge	why-explanations, how-explanations, and cognitive explanations
Case base	why-explanations, how-explanations, and context

Table 2.1: Knowledge containers and their contribution to explanations [Roth-Berghofer, 2004]

The case base contains experience solving prior problems and is defined in terms of the vocabulary.

All these containers contain domain knowledge in some form. The case base in the form of specific cases and the other more generally in the form of what concepts are and how they relate to one another, conceptually as well as in-concept-variation. It should be noted that similarity measures and adaption knowledge seems to necessarily have a strong coupling. As we see it, in order to be able to adapt some features by a case we must be able to tell in what way the attributes are different.

2.5 Mixed initiative / conversational

In order to increase the accuracy and quality of the solutions given by the system, as well as the ability to adapt to novel situations, the system needs a way to reduce the amount of unjustified assumptions and over-generalized knowledge to a minimum. This is one problem with answers given by methods like decision trees and many other machine learning methods. Once they have been trained they are not well suited to face problems with properties that are different from the ones they were trained with. This is also a problem within CBR, when the domain knowledge has been engineered and the similarity measures have been set, the system has no way of taking attributes it has not seen before into account. One way of handling this problem is to let the system ask the user or an expert when it is in doubt. We say that the system is *mixed-initiative* or a *conversational* system. When it encounters a case with an attribute it has never seen, it may ask how it relates to other attributes; what the semantical meaning of it is and so on. It may also be that a retrieved case has a value for an attribute while the problem case is missing this value. It is then possible to ask for information about this attribute, if it is present in the problem or is it irrelevant.

MYCIN was the first expert system where the user could interact with the system by asking for explanations for the answers it gave. The user could type "why" or "how" in order to find out why the resulting rules were used and how they relate to other rules respectively. It was also possible for the user to enter new knowledge into the system by entering new rules. [Shortliffe et al., 1975]

The first CBR-like system that used mixed-initiative was Protos [Porter et al., 1990]. The conversation is started by the user, who usually is an expert, by describing the problem case. Protos then suggests a solution which the user can accept or reject. If the user rejects, Protos explains why it classified as it did and the user is given the opportunity to explain why certain attributes was as important as they where and what classification Protos should have proposed.

Moore and Swartout [1991] identifies three challenges with conversational systems. The system needs to understand the answers and explanations it gives in order to be able to further elaborate on them. By now it should be clear that this is a key problem faced by explanation-aware systems in general. The questions asked need to be interpreted in a way that is context-aware. Early systems like MYCIN always interpreted "why?" as what higher domain goal gives rise to the current one. Different user and different contexts require different ways of presenting explanations. In order to have high explanation-success the system should be able to use several explanatory strategies and methods. This is relevant in both the cases where a particular context needs explanation and when the user needs an explanation on a particular level. The system designed and built by the authors addresses all these challenges in the following ways.

- Plan responses such that the intentional structure of the responses is explicit and can be reasoned about.
- Keep track of conversational context by remembering not only what the user asks, but also the planning process that led to an explanation.
- Taxonomize the types of (follow-up) questions that are asked and understand their relationship to the current context.
- Provide flexible explanation strategies with many and varied plans for achieving a given discourse goal.

Interactive CBR systems are called *conversational case-based reasoning*(CCBR) systems[Aha et al., 1998]. The greatest benefit CCBR has over traditional CBR is that users are not required to initially provide a complete description of their problem. The user starts by describing the problem, then the system assists in further elaborating the problem during a conversation. This process ends when the user accepts a solution to retrieve.

The largest problem within CCBR is *dialog inferencing*, dynamically computing inferences from user input. This involves both interpreting what the user means by a query and inferring what questions are appropriate to ask. It is not necessary to ask a question if it can be inferred from earlier input or from domain knowledge. An example presented in Aha et al. [2001] is in the domain of printer troubleshooting. The system is presented with a problem where the printout has black streaks. A question that does not need to be asked is whether the quality of the printout is good or bad, since it can be implied from the opening query that it is in fact bad. The authors point out that existing solutions use explicit rules to solve this problem, but that this is not manageable due to the daunting task of maintaining these rules. The approach taken by the authors to overcome this is to use model-based reasoning to generate implication rules.

Another issue addressed by Aha et al. [2001] is simplifying case authoring. CCBR cases are usually heterogeneous, the overlap of the sets of questions used to define each case's problem specification is small. This complicates the problem of deciding which questions and cases to present to the user at each point during a conversation. Poor choices in questions will prevent useful diagnosis of the customer's problem, while poor choices for cases will prevent a good solution from being retrieved. They present a list of design guidelines that should be followed in order to design good case libraries. But the authors point out that it may not always be easy to consistently follow all the guidelines, since they may contradict one another and the learning curve for authoring cases is long. The authors also present a method for revising the case library in order to improve the efficiency, defined as the number of questions asked before retrieval occurs, and precision, defined as whether the retrieved case's actions solve a user's query.

Their approach is using a top-down decision tree induction algorithm to restructure the case library based on the most frequently answered question among a node's cases.

Gupta [2001] states that abstraction has not been addressed adequately in CCBR systems and that this leads to the following problems:

- Unwanted correlation among features.
- Limited ability to assess similarity.
- Redundant questions are generated during conversation.
- Loss of decisional information due to feature generalization.
- Difficulty in assigning indices.
- Inconsistencies develop in case representation when new features are added.

To address these, the author make use of integrated methodology called Taxonomic CCBR that uses feature taxonomies for handling abstraction. This means that relevant domain features (e.g., test results) are arranged in feature taxonomies in which levels of abstraction are explicitly represented by subsumption links, thus making a parent's features appear in all cases where its children appear. Their Taxonomic CCBR include the following:

- A set of questions that are used for indexing cases.
- Feature Taxonomies, being acyclic directed graphs comprising nodes of question-answer pairs.
- A set of cases consisting of a problem state and a solution.

When a query is given to the system, questions that matches the problem are identified and the taxonomy is searched in order to find candidate cases. The cases are then ranked by aggregating similarity of question-answer pairs and case description similarity and presented to the user. The authors point out four benefits to this approach. Consistent and efficient representation, accurate and responsive retrieval, responsive conversation with reduced information load, and simplified and flexible case maintenance.

Aha et al. [2001] identifies that since the individual taxonomies were isolated from each other, and from other information sources that could support query elaboration, information is prevented from being propagated to these taxonomies, and could inflate the length of the user's problem-solving session. The author argues that the taxonomic CCBR approach, as originally conceived, captures abstracted relations among features. These are not the only type of inter-feature relations that should be exploited. In particular, exploiting causal and implication relations can also contribute towards query elaboration. The improvement involves incorporating dependency relations among features (e.g., causal, implied, sequential) to improve both representational and query elaboration efficiency.

Gu and Aamodt [2005] points out that the then proposed metrics for question selection, static decision tree; information gain; occurrence frequency; information quality; similarity variance; and attribute-selection strategies; are not good enough since they are knowledge-poor, i.e. only take statistical information into account. The authors identifies that in the tasks feature inferencing, question ranking, question clustering, and question sequencing general domain knowledge has a potential to control and improve the process. They describe a function that maps concepts to questions, where concepts are represented as nodes in a semantic network. This would improve the efficiency and precision of the system, but the authors point out that it does have two weaknesses. Much work is needed to acquire the required knowledge as well as maintaining the domain knowledge and case library to maintain consistency.

Gu and Aamodt [2006] proposes to not only retain the successful cases, but that also the conversation itself should be as well. The purpose of this is to improve the efficiency of CCBR from the perspective of shortening the dialog length. Their framework is illustrated in 2.5.

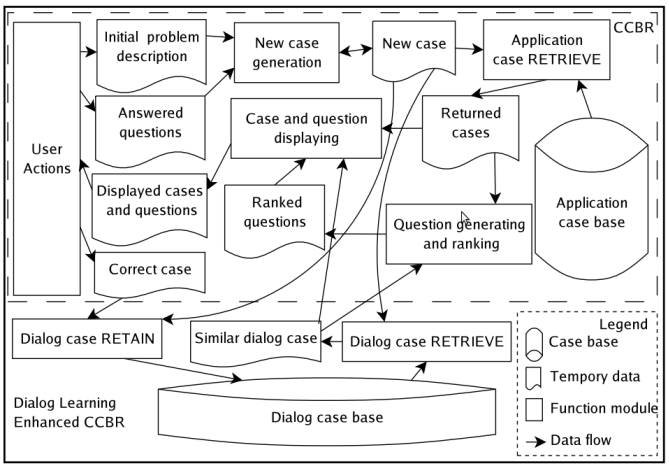


Figure 2.5: A framework for the dialog learning enhanced CCBR [Gu and Aamodt, 2006]

The framework uses a separate case library for dialog cases that is used in order to determine how to respond to the user. For a dialog case, the problem description part contains the information related to the dialog process: the initial constructed new case, the later incrementally selected questions, and their answers. The solution description of a dialog case refers to the case successfully retrieved from the application case base. For all dialog cases, their outcomes are the same, that is, the user gets the retrieved application case, and terminates the dialog. The implemented system shortened the dialogs in 29 of 32 datasets.

2.6 Knowledge representation

In order to be able to reason about a concept and explain it, it needs to be in some way available for us to do so. We need to have an internal representation of the knowledge we seek to reason about and explain. There are two aspects of a knowledge representation language: the syntactic, concerning the way information is stored in an explicit format, and the inferential aspect, concerning the way the explicitly stored information can be used to derive information that is implicit in it [Reichgelt and Shadbolt, 1991, p.3]. Davis et al. [1993] presents five roles knowledge representations have in artificial intelligence systems, and in natural intelligence systems as well, that will now be presented.

A surrogate Meaning that it is an incomplete representation of a concept, the only complete representation is the concept-instantiation itself. This highlights one of the biggest challenges in the AI field, that the real world is quite complex and that we are not close to being able to create a model that is completely accurate and capture a broad range of domains.

A set of ontological commitments This is something that follows from the fact that the representation is a surrogate. Since we are not able to accurately model all aspects of the world, we have to make decisions to represent the knowledge in a way such that some aspects are more accurately represented than others. We have to focus on the aspects of the world knowledge that we deem important, e.g. focusing on the domain of medicine and ignoring printer maintenance. We mentioned this earlier, a broader focus results in a larger knowledge base that requires more resources to reason with.

A fragmentary theory of intelligent reasoning The knowledge representation we commit to does not only dictate how we represent our knowledge, it also dictates what kind of reasoning it is possible to perform. The authors mention five paradigms, mathematical logic, psychology, biology, statistics, and economics. All have different approaches to intelligent reasoning, determining the kinds of inferences possible to make.

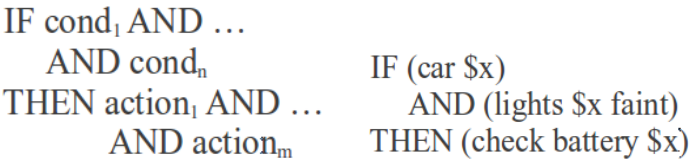
A medium for efficient computation This is tightly coupled with the previous rule. A particular knowledge representation exists because it captures knowledge in a way that both manages to contain some knowledge and it is efficient to do reasoning with it. If this is not the case, there is no point that the representation exists. This may perhaps be an overgeneralized statement, but as long as there does not exist some better representation that both enables us to do inference with the knowledge and is efficient, it is true.

A medium for human expression The knowledge we seek to represent in order to enable the computer to reason with it, is in essence knowledge that humans possess and want to communicate to the computer. Thus it must in some way be understandable to humans, in knowledge engineering and verification.

2.6.1 Production rules

A rule-based system uses rules on the form shown in Figure 2.6, consisting of an IF-part(antecendent) and a THEN-part(consequent). Many systems have been build using production rules, among them MYCIN [Shortliffe et al., 1975] mentioned earlier. One of the advantages in these kinds of systems is the naturalness with

which expert knowledge can be expressed in rules as "rules of thumb" that the expert has made based on experience. It is also easy to divide the rules into natural and intelligible chunks. Since each rule is independent it is easy to update each rule independently. The rule base does however grow quite large as the system is taught new knowledge. A large rule base leads to, among other things, relatively much computation needed to do matching. In terms of explanation these systems are capable of generating transparency explanations, but are for instance not able to explain why a particular rule was matched rather than another. [Reichgelt and Shadbolt, 1991]



(a) The general pattern for production rules

(b) A rule telling to check battery if the lights are faint

Figure 2.6: Examples of production rules

2.6.2 Dynamic memory

Schank [1980] introduces the concept of *dynamic memory* and *memory organization packets*(MOPs). The premise of dynamic memory is that remembering, understanding, experiencing and learning cannot be separated from each other. A MOP is a memory structure that help organize episodic memory, as well as help to process new inputs. New information is stored in terms of the high level structure that was used to interpret it. It contains both general knowledge, and organize specific experiences of this knowledge in a complex hierarchy. The author use visits to the dentist, doctor and lawyer to illustrate how situations are stored. A visit to these offices all have both shared and distinct elements. Since the situation where you wait to get in is shared by all of them, it is organized in a structure that is the same for all situations. When encountering a new situation, it is stored in memory only if it differs significantly from previous experiences. I.e. if it contains anomalies or expectations where violated. The specific experiences, the cases, are in fact indexed primarily by these anomalies. The author claims that understanding means being reminded of the closest prior experience and being able to use the expectations generated by that particular reminding in the new situation.

2.6.3 Semantic net

A semantic network is a network which represents semantic relations among concepts. In most cases it is made up of other concepts mentioned in this section. In the CREEK-architecture the knowledge structure is made of frames that are organized in a semantic network. Semantic nets are primarily based on the intuition of interconnectivity while frames stress the intuition that knowledge should be organized in larger chunks. A network consist of two things, nodes and links. Nodes correspond to objects or classes of objects in the world while the links correspond to the relationship between these objects. In a pure semantic net, the nodes them selves do not contain any information.

2.6.4 Ontologies

An ontology defines the basic terms and relations comprising the vocabulary of a domain, and contains precisely defined terms that can be used to describe and understand more complex descriptions. It is a formal model about how we perceive a domain of interest and provide a precise, logical account of the intended meaning of terms, data structures and other elements modeling the real world.[Flouris et al., 2008] This means that an ontology in fact has some of the properties of semantic nets, since the representation of the different concepts and their relation to one another makes a net. This can be formulated formally as O = (S,A). Here O is the ontology, S is the (ontological) signature, describing the vocabulary, and A is a set of (ontological) axioms, specifying the intended interpretation of the vocabulary in some domain of discourse. [Kalfoglou and Schorlemmer, 2003] I.e. the signature is the entities defined in the ontology and the axioms are the relations that may be defined between the entities.

Ontologies are often implemented in description logic or frames. When description logic is used, the meaning of entities and relations are combined with the well defined semantics for reasoning. Two things that make description logic very suited as a knowledge representation and reasoning tool are the concepts of subsumption and instance recognition. Subsumption determines whether a term is more general than another, and instance recognition finds all the concepts that an entity satisfies. Furthermore, completion mechanisms perform logical consequences like inheritance, combination of restrictions, restriction propagation, contradiction detection, and incoherent term detection. [Díaz-Agudo and González-Calero, 2000]

2.6.5 Protégé

Protégé is an application for editing ontologies and started as the thesis work of Mark Musen. The goal was to reduce the role of the knowledge engineer, thus min-

imizing the knowledge-acquisition bottleneck when constructing knowledge bases. This work was initially just aimed at building knowledge-acquisition tools for a few specialized programs in medical planning, but have since then evolved into a much more general-purpose set of tools with a large community of users and contributors.[Gennari, 2003]

Currently there are two versions of Protégé in active use, Protégé-Frames and Protégé-OWL or Protégé 3 and Protégé 4.[4] These two variations of separating the two versions are not equivalent. Protégé 3 exists in editions handling either frames or OWL respectively, while Protégé 4 is strictly OWL based. The main difference between the two OWL-versions is that the earliest version handles OWL 1.0 (and RDF(S)) while the latest version handles OWL 2.0 and makes use of the OWL API developed by The University Of Manchester [Bechhofer et al., 2003] rather than the custom Protégé OWL-API, which is simply a layer on top of the old frame-based system. The 4.0 version does however not have full support for OWL 2.0, this is introduced in version 4.1 which at the time of writing is in beta.

For developers it is recommended to use the latest version unless RDF-support is needed. From a user-perspective the same is recommended, but the fact that functionality provided by plugins are not necessarily available in the latest version.[Vendetti and Drummond]

2.6.6 Frames

A frame is a data structure that capture a stereotyped situation and has several types of information attached to it, some about the situation and the concepts, and some about how to use the frame [Minsky, 1974]. A frame can be seen as a network of nodes and relations. The top-level of the frame is somewhat constant, it represents what is always true about the supposed situation. The lower level have many terminals, or slots, that may be filled with specific instances or data. This means that a frame may have sub-frames. In addition to the ability to have sub-frames, it is also possible that slot refers to another frame without it being a sub-frame. There are two types of frames, class frames and instance frames. It should come as no surprise that the class frame describes a class of entities in the world, while the instance frames describes an individual of a given class. The advantages frames are claimed to have are several. The frame represent knowledge the way in which domain experts think about their knowledge, the knowledge is structured in the same way the domain is structured. The hierarchical structure provides advantages from both an epistemological and a inference point of view, being both expressive and efficient to infer with. The drawbacks pointed out are that there are no semantics defined for the languages, that poly-inheritance may

[4] http://protege.stanford.edu

lead to problems and that there are some expressive limitations when it comes to incomplete knowledge.

A slot may be one of two types, a *template slot* or an *own slot*. The difference is that a template slot is attached to the class while the own slot is attached to the object represented by the frame. This means that subclasses will only inherit template slots, the own slots are specific for each entity. It is possible to constrain the allowed values of a template slot by adding a *facade* to the slot. This makes it possible to constrain the cardinality, value types, minimum and maximum value for a numeric slot, and so on. In Figure 2.7 we can see a how the concept of a car

```
car
    has-instance            value             car#1
    subclass-of             value             vehicle means-of-transportation sporting-gear
    has-part                value             wheel fuel-system engine electrical-system
    has-number-of-wheels    default           4
    has-colour              value-class         colour
    has-fault               value-class         car-fault
    has-age                 value-dimension   years
                            if-needed         (time-difference *current-year*
                                               self.has-production-year)
```

Figure 2.7: The concept of a car represented in the frame-based language CreekL [Aamodt, 1994]

is represented in the frame-based language CreekL.

There is not a single standard frame representation language. Some of the languages that exists in addition to CreekL are Ontolingua [Farquhar, 1997]; Loom [MacGregor, 1999]; and Protégé-2000 [Grosso et al., 1999], which are all different. The latest version of these languages does however have in common that they support the Open Knowledge Base Connectivity (OKBC) protocol [Chaudhri et al., 1998]. OKBS is a protocol for accessing knowledge bases, thus decreasing the gap between the different languages used to represent frames. Even though it is not a representational language it does assume an implicit representational formalism called `OKBC Knowledge Model`, specifying constraints on the basic primitives of the language. [Wang et al., 2006]

2.6.7 Description logic

Description logic is a family of knowledge representational languages that use formal logic-based semantics to represent domain knowledge in a structured and formally well-understood way. Some description logics include operations, like transitive closure of roles, making it able to infer things that cannot be expressed in first order logic. The reason that it is called description logic is that important concepts of a domain is described using atomic concepts and atomic roles (unary and binary predicates respectively) and that formal, logic-based semantics are

used. [Baader et al., 2008] We will not go into the details of the logics involved, but rather describe how it is used as a knowledge representation by using it to implement ontologies.

2.6.8 The semantic web

The traditional web was not made in such a way that computers easily could interpret the meaning of the content. The data from one application is often understandable to only that application. It is possible to view bank statements, pictures and appointments in a variety of applications, but there is no standard way to relate the different things to one another. This is what the semantic web is about, making a standard way of interpreting the meaning from the data. Then it is easier to make an application view the bank statements and pictures in the calendar, showing when they where executed and taken respectively. And since the meaning of everything is encoded in a standard way it is easy to use the exact same data in any application that is compatible to the standard.

Another example is visiting a profile for a person on `http://idi.ntnu.no/people` or any other such page. For a person it is easy to see what type of information this page contains, a computer however will have to be very complex in order to guess the nature of the page's content and what to do with it. As we

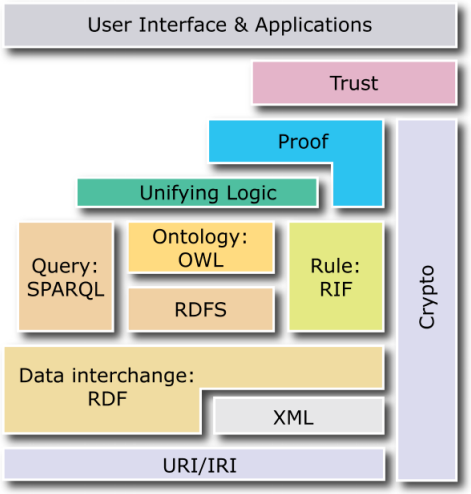


Figure 2.8: The different components of the semantic web. `http://www.w3.org/2001/sw/`

can see in Figure 2.8, `the semantic web` involves many concepts. We will focus on the *OWL* part of the framework.

2.6.9 OWL

OWL is an acronym for *Web Ontology Language* and is a specification for a family of languages intended for authoring ontologies. The specification is endorsed by the World Wide Web Consortium (W3C)[5] and the latest version of it was finalized in October 2009. The purpose of the specification is to define the syntax and semantics of a knowledge representation, specifying terms and their relationship with other terms in the ontology.

An OWL ontology is made up of classes, properties, individuals, and data values. A class may be defined to be have super classes, equivalent classes, disjoint classes and individuals. Individuals in turn have, in addition to membership in one or more classes, *object* and *data* properties. An object property refer to another individual, while a data property holds a primitive value such as an integer or string. These properties can, just as classes, be organized in a hierarchy. Properties may also have a *domain* and a *range*, specifying the values that can have the particular property and the values the property may have respectively. Object properties may also specify that they are *functional*, *transitive*. *symmetric*, and *reflexive* as well as their inverse.

Species

Both the first and second (current) version of OWL were defined in three variants with different level of expressiveness, thus also different level of computational complexity. The first version had the variants `Lite`, `DL` and `Full`. Where Lite is the least expressive and Full is the most expressive. DL is in the middle of these two, designed to be as expressive as possible and also be computational complete, decidable, and compatible with the reasoning algorithms available. It has this name because description logic is what is used to do the reasoning. [McGuinness et al., 2004]

While the variants of the first specification each were a syntactic extension of its predecessor, the variants of OWL2, or profiles, are sub-languages of the OWL2 language that offer advantages in particular application scenarios. [Group, 2009]

OWL 2 EL enables polynomial time algorithms for all the standard reasoning tasks; it is particularly suitable for applications where very large ontologies are needed, and where expressive power can be traded for performance guarantees.

OWL 2 QL enables conjunctive queries to be answered in LogSpace using standard relational database technology; it is particularly suitable for applications where relatively lightweight ontologies are used to organize large num-

[5] `http://www.w3.org`

bers of individuals and where it is useful or necessary to access the data directly via relational queries (e.g., SQL).

OWL 2 RL enables the implementation of polynomial time reasoning algorithms using rule-extended database technologies operating directly on RDF triples; it is particularly suitable for applications where relatively lightweight ontologies are used to organize large numbers of individuals and where it is useful or necessary to operate directly on data in the form of RDF triples.

2.6.10 Frames vs OWL

Wang et al. [2006] compares the paradigms of frames and OWL, highlighting the similarities and differences. They are both focused on classes that have properties and instances, representing a concept in the domain in question, but the semantics of the modeling constructs are different.

When two objects have the same name in frames they are assumed to be the same object, while the name of objects do not matter in OWL, where it has to be stated that they are the same object. This is however not true in the practical sense. All entities in OWL has to have an unique identifier, URI, which in most cases consists of the ontology identifier and the name of the entity.

Frames assumes a closed world with negation as failure, everything is prohibited unless it is explicitly stated that is permitted and if a fact is not present and cannot be proven it is assumed to be false. OWL on the other hand assume a open world with negation as unsatisfiability, everything is permitted until it is stated that it is prohibited and a statement is false only if it contradicts other information in the ontology. From this follows that in frames you cannot enter something into the system if there is not a place for it in the corresponding template, while anything can be entered into OWL unless it violates one of the existing constraints. In frames, a person is not allowed to have a jacket unless the frame has a jacket-slot, but this is possible in OWL.

A frame ontology only has one model which is the minimal model that satisfies each of the assertions of the frame ontology. This means that models for a frame ontology can only contain instances that are explicitly specified. In general an OWL ontology will have many models consisting of all possible interpretations that satisfy each of the assertions in the OWL ontology. In Frames, defining facets on a slot in a class, or defining a constraint on a slot at the top level, makes a statement about all instances of that class (except for possible exceptions provided by default values), describing necessary conditions for instances of that class.

Statements about OWL classes can be of two kinds, properties that are true for all individuals of that particular class; and properties that are collectively necessary and sufficient to recognize members of a class. An OWL classifier can

use the sufficient conditions to infer which classes are subclasses of the defined class.

When reasoning with OWL ontologies, a model that satisfies all the axioms in the ontology is built. If this is not possible, the ontology is inconsistent. By contrast, a Frames reasoner checks if the constraints are satisfied by the property values on instances; if they are not, the instance is said to be non-conformant.

In practice, these major differences in the sanctioned inference lead to differences in modeling style. A developer of an OWL ontology thinks in terms of necessary and sufficient conditions to define a class, building new concepts from existing ones by fitting them together in definitions like blocks of Lego and determining what conditions sufficiently define something as an instance of a class. A developer of a Frames ontology addresses the problem from another angle, deciding what the implications of being a member of a particular class are.

In frames all subclass relations and instantiation relations has to be asserted explicitly. When we state that a vegetarian pizza is a pizza with exclusively vegetables as topping and that mushroom pizza has only mushrooms, a frame reasoner does not know that the pizza is a `Vegetarian pizza` until we explicitly state this fact. A OWL reasoner on the other hand can infer that the pizza is a vegetarian pizza since the fact the mushrooms are vegetables. In OWL it is possible that an instance becomes member of several classes as a result of reasoning. If we for instance define that `MargheritaPizza` (which is topped by only tomato sauce and mozzarella) is of the class `Pizza`, it may be classified as `Vegetarian pizza` as well given that we have a statement saying that all pizzas without meat is an vegetarian pizza. In Frames however, it is not possible for the inference to assign new classes to an instance. A typical reasoning method in frames is constraint checking, where a reasoner determines whether slot values for instances of a class satisfy the constraints defined for the class. In OWL, a reasoner performs consistency checking, determining if there is at least one model that satisfies all the assertions in the ontology. This should not come as a surprise due to the use of a closed world assumption and a open world assumption respectively.

The authors concludes that frames are appropriate to use where the closed world assumption is appropriate; there is a focus on data acquisition; constraints on slot values are required; and the model relates classes to other classes. The use of OWL is appropriate when creating robust terminologies in which classes are defined; there is a need for reasoning to ensure logical consistency of ontologies; terminologies are published on the semantic web and accessed by other applications; and where classification is a paradigm for reasoning.

It does however seem like the community is abandoning Frames in favor of OWL. Given the background we have presented this is quite natural, since OWL has a clear specification, formal semantics and a well defined role in `the Semantic`

`web`. A sign indicating this is that while Protégé 3 existed in two versions, -Frames and -OWL, Protégé 4 exist only in an OWL version.

2.7 Merging ontologies

One of the benefits of ontologies in the semantic web is that knowledge can be separated into separate ontologies according to areas of concern. It is natural that knowledge about medicine is contained in a different ontology than an ontology containing knowledge about mechanical engineering. These two ontologies may or may not be made by different people, be hosted on different hosts and there may not be a single person that is aware that both ontologies exist.

It may also be that someone is starting a project and does in fact need knowledge about both of these subjects. If this is the case, they may benefit from merging these two ontologies into a single ontology such that the concepts in the two are linked to each other. The subject of ontology merging is on the border of the scope of this project, but is in fact relevant for the generation of test data when evaluating our work. Because of this we will briefly present the different terminology and challenges within this subject.

The terminology used regarding this subject has to a certain degree been somewhat unclear and confusing. This is because there are many different aspects of the problem of how ontologies may change, resulting in several interlinked and partly overlapping research disciplines which may in some cases use different terminology. We will now present the most important terms used withing this area based on the survey performed by [Flouris et al., 2008].

2.7.1 Ontology mapping

The result of a mapping is a function mapping the entities(vocabulary) and axioms in one ontology to the corresponding in a different ontology. This function can be be either one-way (injective) or two-way (bijective) and either a total mapping or partial mapping. An example is having two ontologies concerning food where different names are used for the entities. A mapping can then be created to relate the two. If the two ontologies contain exactly the same entities this is a total mapping, else it is a partial mapping.

Ontology morphism Refer to the activity of creating these functions, or morphisms, relating the ontologies. The difference between ontology morphism and *ontology matching* is that a morphism relate both the entities in the ontologies as well as the axioms. I.e. when only the entities are being matched it is called

ontology matching and the result of this is called an *ontology alignment*. Due to this the terms matching and alignment are used interchangeably.

2.7.2 Ontology evolution

As our understanding of a subject changes, the domain changes, requirements change, we may need to change a particular ontology. Ontology evolution is just this, that the ontology changes as the environment it is supposed to model changes. *Ontology versioning* refers to the ability to handle these changes. An ontology may be used by some other resource, and because of various reasons it is an advantage to know which version of the ontology one is depending on.

2.7.3 Ontology integration and merging

The activities ontology integration and merging refer to roughly the same thing, namely creating a new ontology based on the information in two ore more source ontologies. It does however seem like it is these concepts that has given rise to the most confusion around their meaning. The authors does however conclude that the term `ontology integration` refer to the process of combining ontologies about similar, but not identical domains, into one ontology. `Ontology merging` on the other hand is the process of combining ontologies about identical domains into one ontology. The distinction is illustrated in Figures 2.9 and 2.10.

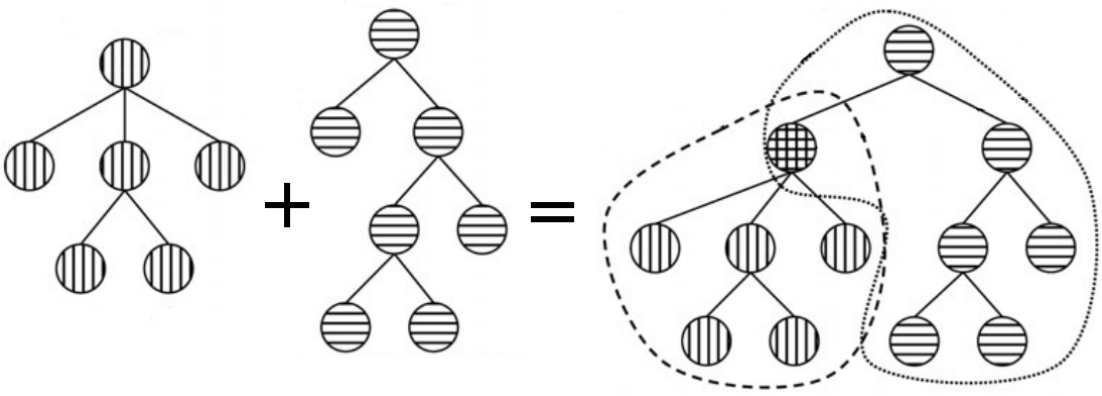


Figure 2.9: When the ontologies are similar, but different domains, they are integrated. It is easy to see which part came from which source ontology. Adapted from [Flouris et al., 2008]

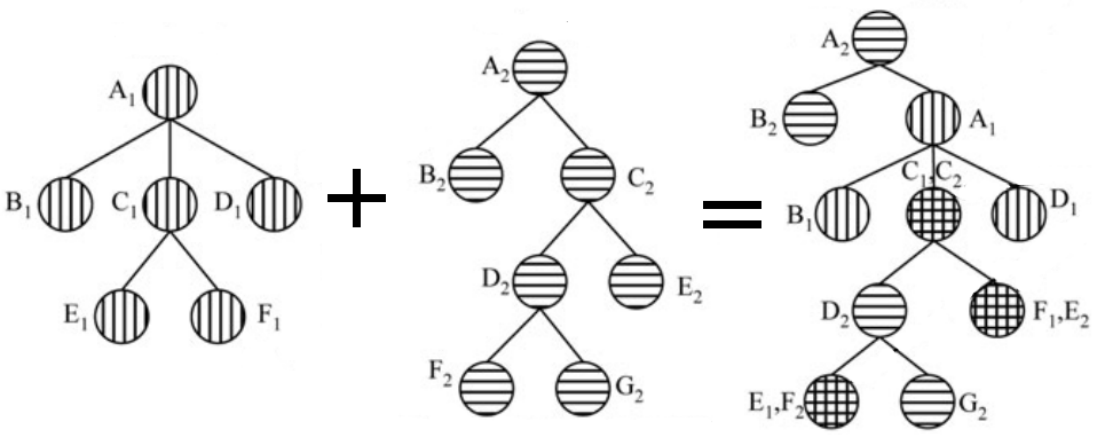


Figure 2.10: When the ontologies are about identical domains, they are merged. It is difficult to identify which part came from which source ontology. Adapted from [Flouris et al., 2008]

2.7.4 Merge vs import

There are two ways of creating a new ontology from other existing ontologies, merging all of them into one file or simply importing the existing ontologies into the new ontology. The distinction is that in the former approach all the assertions from the existing ontologies are copied into the new ontology, while in the latter the existing ontologies are only referenced in the new ontology.

As we have understood it the use of "merge" is different in the notion we just mentioned and as it is used earlier in the section. It is our understanding that the material described earlier did not care whether the assertions was duplicated or not, it was just concerned by how the assertions in the different ontologies relate to one another.

The distinction is quite important since smaller ontologies naturally are easier to maintain and when assertions from an existing ontology are copied into a new ontology there are no way of keeping these up to date with the changes that may or may be done in the original ontology.

Chapter 3

My Explanation-Aware Case-Based reasoner

Due to the emergence of `the semantic web` and the role OWL has as a knowledge representation it is clear that both are going to have a significant role in the artificial intelligence community in the coming years. Because of this we have created a plugin to Protégé 4 that stores all its knowledge in an ontology, as well as using the ontology when retrieving cases. The two main components in this application are Protégé 4.1 and myCBR 3.0, both having beta status.

The reason for basing our implementation on this particular version of Protégé is not only that it is the latest version, but also that full OWL 2.0 support was introduced in this version. Protégé 4.0 does not have complete OWL 2.0 support, due to the fact that it uses version 2 of the OWL API[1] while full OWL 2.0 support was introduced in version 3 of the API. This does however not mean that there are any problems using this version of Protégé, only that the developers want users to start using it and make sure there are no significant bugs before releasing the final version. The motivation for using myCBR 3 is that the previous version was based on Frames and that there exist no version of myCBR for Protégé 4.x.

In the following sections we will present the structure of a plugin for Protégé 4; myCBR 3; and our implementation, my Explanation-Aware CBR (myEACBR). We will naturally present the capabilities of our implementation as well. The goal of myEACBR is to create an application that is able to explain its actions and answers, fulfilling as many of the criteria described in section 2.1.2 and 2.1.3 as possible. Unfortunately we have, due to limited time and resources, only been able to implement a system that provides explanations for the retrieval of cases, leaving the steps reuse, repair and retain unimplemented. Due to this, our system is not really a CBR system. We will however keep referring to it as a CBR system

[1] `http://owlapi.sourceforge.net/`

since we have referred to it as this up to this point, and we intended to implement all four steps when we started this project.

In the process of developing, we have made some changes to the myCBR 3 code. These changes and the reason for them will be presented in this section.

3.1 Protégé plugin

Protégé 4 has been developed with a strong focus on being modular. It is encouraged to develop all new functionality as plugins. To support the modularity goal the OSGi framework[2][Alliance, 2007] has been used as the core plugin infrastructure, resulting in that all plugins are executed completely isolated from all other plugins and are only aware of the functionality offered by the API provided by Protégé. By implementing the correct interfaces and extending the correct classes it is possible to add tabs, renderers, views and many other both visible and background components. All description logic reasoners are for instance implemented as plugins. [3]

3.2 myCBR overview

myCBR 3 is, as mentioned earlier, independent from Protégé and any other applications. The architecture is centered around the concept of an attribute; attribute descriptions; ranges and concepts, as shown in Figure 3.1. This section will present myCBR as it was before we made any changes to the code. In section 3.3 we will present the changes that has been performed.

An *Attribute* can be anything, there are subclasses for `String`, `Date`, `Number`, and several more. Each Attribute has a distinctive name and a reference to an *AttributeDescription*. The AttributeDescription is used to link the Attribute to other Attributes, as well as `Instances`. It also has a reference to a *Concept*, representing the taxonomic meaning of the AttributeDescription, i.e. the meaning of the link between the Attribute and Instance (or another Attribute). An example is that an Instance has an attribute representing the number of persons attending a dinner. The Instance then has a reference to an AttributeDescription that has the name `numberOfPersonsAttending` and a Concept with this meaning. The Instance also has a reference to an Attribute of a nummeric type and the value of the number of persons, which is associated with this AttributeDescription internally in the Instance.

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

[3] `http://protegewiki.stanford.edu/wiki/PluginTypes`,`http://protegewiki.stanford.edu/wiki/PluginAnatomy`,`http://protegewiki.stanford.edu/wiki/P4APIOverview`

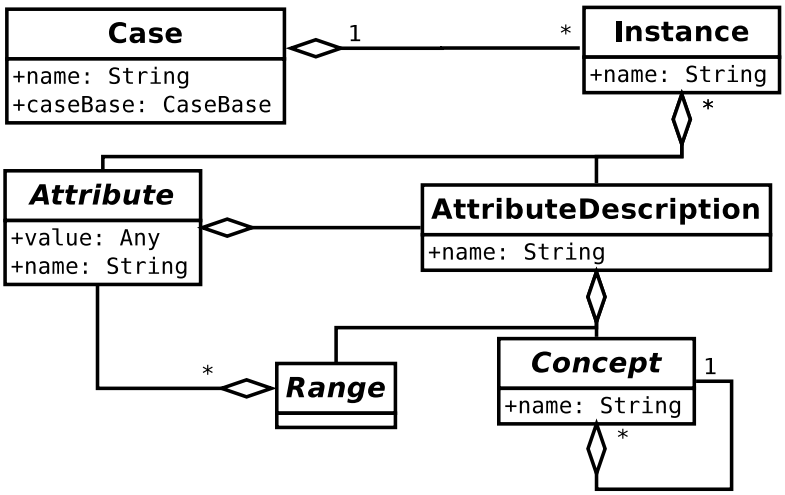


Figure 3.1: Core concepts in myCBR 3

Concepts each have a name and may be arranged in a hierarchy, each Concept having one parent and zero or more children. An *Instance* represents an instance of a case in the case base and has Attributes and AttributeDescriptions, which naturally represents the attribute value for the case. The instance has reference to zero or more *Case*s, that has references to the author of the case as well as the case base that contains it. The fact that the `Instance` class inherits the `Attribute` class indicates the motivation for splitting the responsibility of the CBR-case into the two classes Instance and Case. By doing this the `Instance` can be used as an attribute itself, representing objects that are composed of multiple attributes. In addition to a reference to the attribute value, the AttributeDescription also has a reference to the *Range* of the Attribute. The Range has references to all the Attributes that have the same meaning. I.e. if there are two cases each having a different number of attending persons, both cases will refer to the same AttributeDescription with name `numberOfPersonsAttending`, but to different Attributes. These two Attributes then make up the `Range` of the AttributeDescription, keeping track of all the different values that this AttributeDescription has.

Other key classes are *ConceptExplanation*, *Similarity* and *SimilarityFunction* (Originally an interface called `ISimFct`. We will come back to this kind of name in the next section) ConceptExplanation is a simple class, comprising a description, a set of strings meant to represent links and a reference to the explained object. To use it one manually has to construct explanations and possibly put them into the *ExplanationManager* which is a simple container one can put explanations into. (It does not save, or do anything else, it simply holds references to the explanations in a map.)

Similarity is a class that represents the results of a measurement of the similar-

ity of two Attributes. It has two fields holding the similarity value, a floating point value between 0.0 and 1.0, one for the exact value and one for the value rounded to two decimals. In addition to this it has a static[4] map having the similarity value as key and the Similarity object as value. When similarity values are calculated in the similarity functions this map was consulted in order to check whether a `Similarity` for this value is cached, if it is not one is created. There is no reference to which two Attributes have been compared, and there is no indication to the motivation for caching the Similarity objects.

It is the responsibility of the implementation of `SimilarityFunction` to calculate how similar two given attributes are. For each attribute type (extension of the generic Attribute class) there should be an implementation of SimilarityFunction handling the attribute. Similarity of StringAttributes is for instance calculated by the `StringFunction`. A function may have configuration parameters. StringFunction has a parameter that determines whether equality (1.0 if the strings are equal, 0.0 otherwise) or an n-gram approach should be used when comparing attributes.

3.3 Contributions to myCBR

> Any fool can write code that a computer can understand. Good programmers write code that humans can understand. – Martin Fowler

It has been thoroughly established that code quality is a critical factor in a software product's success.[Martin, 2008, Hunt, A. and Thomas, 2000, Fowler and Beck, 1999] When starting to develop our application, it became clear that myCBR 3 has some issues in this regard.

3.3.1 Meaningful names

As we started working on the implementation, using myCBR 3 as one of the core libraries, it soon became evident that some changes had to be done. The issue which is most easily spotted is that the names given to classes and variables are in many cases strange. An example already mentioned is `ISimFct`, the interface that defines the methods that similarity functions have to implement. There are two errors in this name. One is that the first letter is `I` to indicate that it is an interface. This is a fairly common anti-pattern that should be avoided. The fact that a particular class is an interface does not matter for the user of the object, it only matters that it is a SimilarityFunction[Martin, 2008, Ch.2].

[4] In Java a static field has the same value for all instances of the class. `http://download.oracle.com/javase/tutorial/java/javaOO/classvars.html`

The other problem with the name is that, instead of `SimilarityFunction`, it is called `SimFct`. This abbreviation make the code harder to read and impossible to pronounce without sounding like an idiot. For code to be easy to maintain, it needs to be easy to understand what it tries to do. The name of a variable, function, or class, should answer all the big questions. It should tell you why it exists, what it does, and how it is used. If a name requires a comment, then the name does not reveal its intent.[Martin, 2008, Ch.2]

3.3.2 Comments

Most of the code in myCBR 3 suffers from the problem just described, names are shortened and comments are added to try to explain them. It is a myth that good code has lots of comments. In addition to the reason just mentioned, that names should reveal intent, comments also duplicate knowledge. This means that it is necessary to change both the code it self and the comments, but comments are often forgotten and inevitably get out of date. Out of date comments are untrustworthy, which is worse than no comments at all.[Hunt, A. and Thomas, 2000, Ch.2]

Naturally not all comments are bad, but comments that can be replaced by cleaner code are. Fowler and Beck [1999, Ch. 3] point out that comments often are used as deodorant, it is there because the code smells bad. Martin [2008, Ch.4] states that comments are at best a necessary evil, and that the proper user of comments is to compensate for our failure to express ourself in code.

Another type of comments that are bad, and found throughout myCBR, is the `TODO`-comment. It is there because the programmer has identified something that should be changed or extended in the code. An example of this can be seen in Figure 3.2. Rather than making such comments one should have a system for archiving such ideas, as well as bugs found. It does not have to be any more advanced than a file which is maintained, or perhaps a specialized bug tracking system.

```
public Case(.., final String name) throws Exception {
    ..
    this.id = name; // TODO unique!
    ..
}
```

Figure 3.2: A TODO comment indicating that the programmer thinks there should be checks for the uniqueness of the id.

```
..
// delete old desc
for (AttributeDesc desc : oldDescs) {
    attributes.remove(desc);
}
..
```

Figure 3.3: To clarify the intent of these three lines, a comment has been added.

```
..
deleteOldDescriptions(oldDescriptions);
..
```

Figure 3.4: The same intent can be communicated by moving the lines into a separate method with a descriptive name.

Figure 3.3 is an example of code that tries to clarify the intention of iterating through the attributes by using a comment. The code shown can be moved to a separate method and the lines shown in Figure 3.3 replaced with the line shown in Figure 3.4, clarifying the intent of the code while shortening the length of the method.

We have gone through the code and refactored the code such that all names have descriptive names and unnecessary comments are removed. Other examples of this kind of names are `AttributeDesc` and `IntegerFct` which have been renamed to AttributeDescription and IntegerFunction respectively.

3.3.3 Code duplication

In the classes that handled numbers, IntegerAttribute and FloatAttribute as well as the -AttributeDescription; -Function and -Range classes for these, there where code duplication. These classes contained practically the same code except the type of the number value they handled. To improve this a class handling all subclasses of `Number`[5] was created. To create a class handling a specific type of number this class has to be subclassed, specifying the type of number wanted.

There where also some duplication in the -Attribute, -AttributeDescription and -Range classes. In many cases a field for the value handled, as well as reference to the corresponding AttributeDescription, was specified in each of the classes. E.g. a field for the Date in DateAttribute, a field for the string in StringAttribute, and similar in the description and range classes. This has been improved by making

[5]java.lang.Number, the class all numeric classes inherits.

the root class have a reference to the value, and make the subclasses specify the type by using generics.[Bracha, 2004]

3.3.4 Usage of final

Many classes, such as `Instance`; `Case` and `ConceptRange`, where declared as `final`. This means that the classes (or methods marked as final) can not be overridden to change the behaviour of the class. We fail to identify the motivation behind this decision since there are no comments, or any other hints, clarifying the reason these classes *must* be leaf nodes in the class hierarchy.

Another recurring unnecessary use of final is when used for method parameters for simple `set/add methods`, as shown in Figure 3.5. When used in this way the final keyword makes sure that the object referred to by the variable always is the same within the scope of the method. For it to be necessary to use the final keyword on method variables as in Figure 3.5, an anonymous class would have to be declared within the scope of the method.

Figure 3.5 also illustrates the point made earlier about names of variables. Unless it was stated in the javadoc belonging to the method it would be hard to know what the attribute *s* really is. The variable should have the name `attributeValue`.

```
public boolean addAttribute(final String name, final String s){
    return instance.addAttribute(name, s);
}
```

Figure 3.5: Method parameters are needlessly marked with the `final` keyword.

3.3.5 Enhanced explanation support

It is obvious that explanation has been a concern when creating myCBR 3, but it is clear that it either is not complete, or that explanations have been forgotten somewhere along the way. As mentioned in section 3.2, myCBR 3 does not have the same explanation capabilities as the prior version. To obtain `transparency` and `justification` explanations, referred to as backward explanations in [Roth-Berghofer and Bahls, 2008], we have changed the `Similarity` class as well as the `SimilarityFunction` classes slightly.

The original Similarity class was exclusively a container for a floating point number representing the similarity of two objects, it did not contain information about how this value had been computed or even which two objects have had their similarity measured. To improve this we have extended the Similarity class to keep

a reference to both the compared objects, the similarity function used to calculate the similarity value, and a description that may be used to further explain the measurement.

The similarity functions have in addition to being modified to make use of this change, also been given the ability to provide an explanation for how they measure similarity. Each similarity function has a corresponding properties file containing the explanation of the method used. If the function has any parameters influencing the measured similarity, these should be included in the explanation given. This is however the responsibility of the implementer of the similarity function. This is, except the explanation of the similarity function, equivalent to the capabilities of myCBR 2, creating a tree of similarities for each pair of compared objects.

3.3.6 Delegation and instanceof

```
public Similarity calculateSimilarity(Attribute a1,
                                       Attribute a2) .. {
    Similarity res = Similarity.INVALID_SIM;
    if (a1 instanceof SpecialAttribute
        || a2 instanceof SpecialAttribute) {
       res = prj.calculateSpecialSimilarity(a1, a2);
    } else if (a1 instanceof MultipleAttribute<?> &&
               a2 instanceof MultipleAttribute<?>) {
       res = prj.calculateMultipleAttributeSimilarity(this,
      ((MultipleAttribute<?>)a1), (MultipleAttribute<?>)a2);
    } else if ((a1 instanceof SimpleAttribute) &&
                    (a2 instanceof SimpleAttribute)) {
       SimpleAttribute value1 = (SimpleAttribute)a1;
       SimpleAttribute value2 = (SimpleAttribute)a2;
       ..
    }
    return res;
}
```

Figure 3.6: Most of the similarity functions check the class of the given attributes and delegate to other functions. This example is from AdvancedIntegerFunction.

Code such as the one shown in Figure 3.6 is common in the similarity functions in myCBR. The similarity to potentially be returned is initialized to the `Invalid Similarity`, a similarity with value 0.0, which is returned if none of the

defined conditions occur. If the attributes are of some special type, the similarity computation is delegated to a function defined in the `Project` the similarity function belongs to, or computed in the similarity function it self. The value handled by the function in Figure 3.6 is `IntegerAttribute` (which used to inherit SimpleAttribute, in the lines not showing in the figure value1 and value2 is cast to IntegerAttribute).

To improve this we have, as in many other parts of the code, introduced generics to the similarity function. Each of the classes implementing a similarity function may specify which attribute type to handle, as shown in Figure 3.7.

```
public class StringFunction extends SimilarityFunction<StringAttribute> {
    ..
    public Similarity calculateSimilarity(StringAttribute query,
                        StringAttribute instance) {
        ..
    }
    ..
}
```

Figure 3.7: StringFunction extends SimilarityFunction and specifies that it handles only StringAttribute.

Which similarity function should be used to compare a particular attribute, as well as how to weight the result, is determined in the `active amalgamation function` defined on the concept the attribute belong to. We find the way the similarity functions are resolved cumbersome, and have simplified it slightly.

We have moved the responsibility of keeping track of the similarity function to use, and how to weight the calculated similarity, to the AttributeDescription corresponding to each of the attribute types. For all attribute types defined, there already exists a similarity function. Because of this we have defined that each of the different AttributeDescriptions have a default similarity function. The idea is that, when the query is defined it should be possible to define the similarity function that should be used for that attribute, as well as its weight, in an easy manner. The type of similarity function is naturally restricted by generics, so that only the appropriate type is possible to specify for the attribute description. This is better than the original approach, doing an check of the attribute class and returning the invalid similarity when the attribute was of an unsupported class.

By doing this the process of determining which function to use for calculating the similarity is easier, making it easier to introduce new similarity measures. In the old approach the process of determining how to decompose attributes into sub attributes, e.g. decomposing an `Instance`, was cumbersome. This is now up to the

similarity function for the given attribute, and since the desired similarity function for each of the sub attributes are available through its attribute description, the decomposition is trivial.

3.4 Overview myEACBR

We will now present what our implementation is capable of, what elements it contains, and how it is structured.

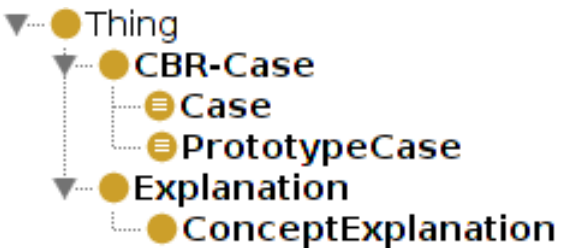
3.4.1 Plugin scaffolding

Protégé is, as mentioned in 3.1, based on the idea that as much functionality as possible should be implemented as plugins. There are a small number of classes and interfaces one can extend or implement and then inject into Protégé by some XML. The "plugin part" of our implementation is defined in three files.

The only source file we have in this regard is *RetrievalComponent* which extends `AbstractOWLViewComponent`. AbstractOWLViewComponent is roughly equivalent to JPanel in Swing[6], it simply allows one to add components to be viewed on the screen. RetrievalComponent contains all of our components, both the ones that are shown on screen as well as the "backend" objects.

When our plugin is initialized by Protégé, through the `initialiseOWLView` method, we obtain a reference to an OWLModelManager, the active ontology, and OWLReasoningManager. After this we make sure that we have a proper reasoner available. By default the reasoner is a `NoopReasoner`, a reasoner which does absolutely nothing. Reasoners are provided as plugins, and we have the Pellet reasoner installed in our installation of Protégé[7]. We have not done a thorough comparison of the different reasoners, but in the small tests performed, the Pellet reasoner used less time finishing the reasoning task.

In order to make the plugin aware of the component, the file *plugin.xml* is needed. This specifies all the components the plugin is made up of. In our case we wanted to have our component in a separate tab. The tab then had to be defined in plugin.xml referring to an XML file specifying which components should be in the tab by default, as well as how they should be positioned.



(a) CBR ontology

CBR–Case
and (isPrototype value false)

(b) Definition of CBR-Case

CBR–Case
and (isPrototype value true)

(c) Definition of Prototype-Case

Figure 3.8: Definition of CBR ontology classes

3.4.2 CBR Ontology

Representation of CBR cases in OWL has been implemented by constructing the small ontology, with namespace `http://www.folk.ntnu.no/lillehau/ontologies/cbr.owl`, shown in Figure 3.8(a). It is made up of the class *CBR-case* having two *defined* sub classes, `Case` and `PrototypeCase`.

There are two types of classes, *defined* and *primitive* classes. The distinction between the two types are that defined classes have both necessary and sufficient conditions, while primitive have only necessary conditions. Meaning that a primitive case is defined in terms of that CBR-Case is its super class, while the cases are defined in terms of that they are equivalent the classes that satisfy the conditions declared in Figures 3.8(b) and 3.8(c).

Case and PrototypeCase

The purpose of PrototypeCase is that it should be a prototype for a particular type of case, defining an attribute that often occur in that kind of case. Attributes for who attended a dinner, what kind of food was served and so on are for instance usually part of a DinnerCase, and it would be useful if these attributes are shown

[6]The GUI framework for Java desktop applications

[7]Pellet is available on `http://clarkparsia.com/pellet`. Other reasoner we have found are available on `http://www.hermit-reasoner.com` and `http://owl.man.ac.uk/factplusplus` as well as the plugin overview in Protégé

by default when either querying for a DinnerCase or creating a new one. This feature has however not been implemented.

Initially we wanted instances not having the isPrototype property to be classified as Case, but we learned that this is not easily achieved. This is because OWL makes the open-world assumption, that there could be other facts that it does not know about as long as they do not contradict what it already knows. The reasoner will not infer anything that could be false in light of those additional facts. This means that all cases have to have the property isPrototype in order to be classified as *Case*.

ConceptExplanation

After an explanation is constructed for a concept we wish to save it so that the process of constructing the explanation does not have to be repeated. We will get back to how we implement the construction of concept explanation and how they are saved and retrieved in section 3.4.9. A concept has the following data properties:

hasConceptName The distinctive name of the concept explained.

hasExplanationSource The full name of the class that constructed the explanation.

hasLink Link to an internet resource used when constructing the explanation.

hasTextualExplanation The explanation text it self.

3.4.3 Dinner ontology

To test our implementation we have chosen the domain of choosing ones dinner. We have created an ontology importing the CBR ontology introduced in the previous section and a wine and food ontology provided by W3C[8]. The two ontologies have been imported into a new ontology with the namespace `http://www.folk.ntnu.no/lillehau/ontologies/dinnerImported.owl`.

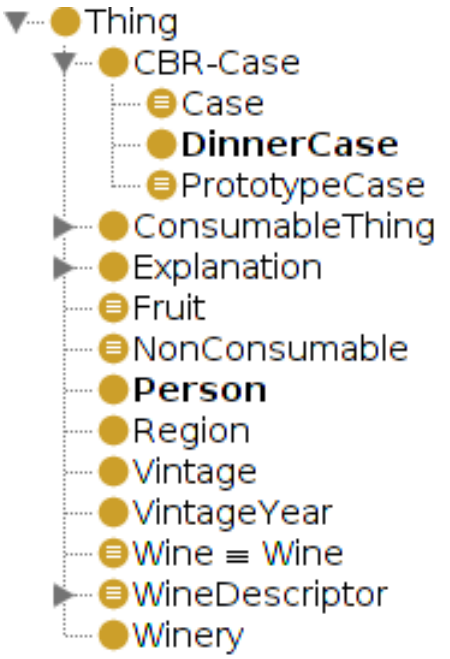


Figure 3.9: Dinner ontology, the class *NonConsumable* is originally named NonConsumableThing but has been renamed to make this Figure narrower.

The top-level of the resulting ontology is shown in Figure 3.9 In addition to these two imports the ontology contains two primitive classes, *DinnerCase*, as a sub class of CBR-Case, and *Person*, as a sub class of Thing. The classes that are defined in the current ontology, not an imported ontology, are show in bold text in the Figure.

Instances of DinnerCase is supposed to be cases of dinners that have been recorded, and instances of Person are normal persons that for instance have eaten a dinner. An instance of DinnerCase typically has data properties and object properties representing the person(s) that has eaten the dinner, where it was eaten, what was eaten, the type of beverages and so on.

As mentioned in section 3.4.2, the most common attributes for a particular type of case is supposed to be defined in a prototype case. But as highlighted in section 2.6.10 an instance of a class can have any property unless it is defined that there should be any restrictions.

3.4.4 Instance attributes

As mentioned in section 3.3 the class representing instances of cases can also be used as an attribute. In the context of OWL this is quite natural because of object properties, where a instance is referred to in an attribute.

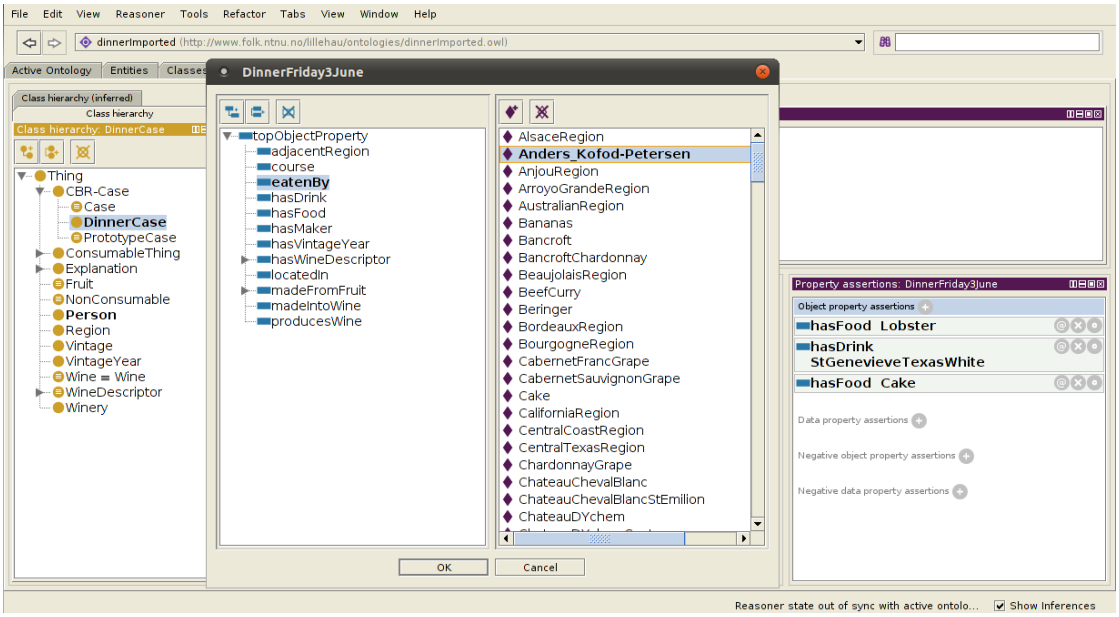


Figure 3.10: The process of creating cases are done in the Protégé ontology editor.

3.4.5 Defining cases

We chose to create cases directly in Protégé instead of creating a separate interface for this in our plugin. This is done by creating instances of DinnerCase and assigning properties to the instance. Figure 3.10 shows a dialogue where the person eating the dinner is being assigned to it. The case already contains the information that lobster and cake was eaten and the wine "St. Genevieve Texas White" was drunk.

3.4.6 Defining queries

Before the user has added any attributes to the query it has attributes for the ID, always being "a query"; the author, defaulting to the system user name; and the time when the query was initialized. In figure 3.11 a query has been created and two attributes have been added. When pressing *New attribute* the process of adding an attribute to the query is started. The first thing to specify is what type of attribute to add. It is possible to select any of the attribute types that are defined in myCBR. This is done by selecting one of the corresponding "radio" buttons, as seen in Figure 3.12(a).

The next thing to specify is what concept the property should belong to, done by selecting a class from the class hierarchy,as shown in Figure 3.12(b). If the

[8] `http://www.w3.org/TR/owl-guide/wine.rdf`

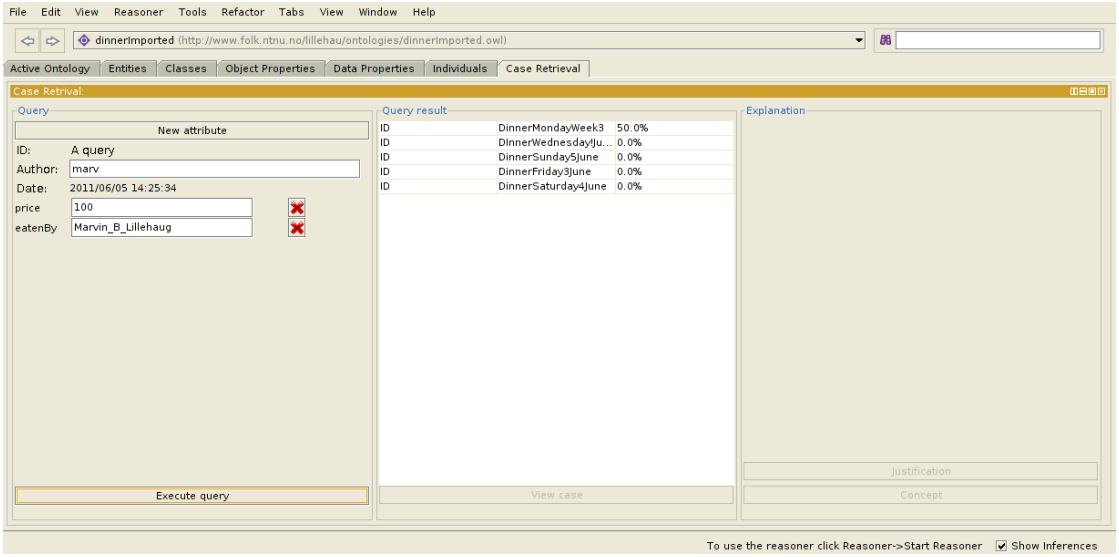


Figure 3.11: An query with two attributes are executed

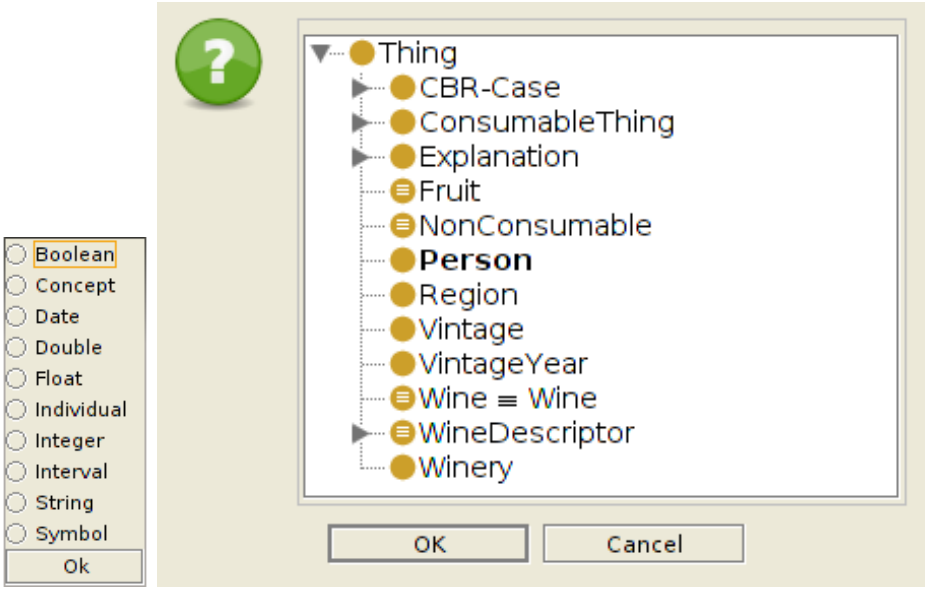
attribute is supposed to represent the person eating the dinner, Person is the appropriate class to choose in this step. After this the name of the attribute has to be specified in a standard input field. To represent the person eating dinner this has to be "eatenBy", as this is the name we have used in all our test cases.

What happens after the name has been specified depends on the type of attribute specified. For most attribute types a standard input field is used to input the desired value. This dialogue accepts only certain input based on attribute type, so that if an invalid value(e.g. "food" in a field for an integer attribute) is specified the input field reappears. The input field is shown until either the value is valid or "cancel" is pressed. When attribute type *Instance* is selected a dialogue showing all defined instances of the specified concept is shown.

When the query is fully specified "Execute query" is pressed, and the query is executed. The top queries are then listed in the middle of the screen ordered by their similarity with the query.

3.4.7 Protégé explanations

It is hopefully clear by now that the main focus of this project is generation of explanations by computer systems. Before we present the explanations provided by our plugin, we will present the explanations Protégé provides for the statements it has inferred. The class affiliations before and after applying a reasoner to the ontology are in many cases quite different. Because of the axioms in the ontology classes may be inferred to be super or sub classes of some other class or it may be inferred that instances are member classes other than the ones asserted.



(a) Choose what type the new attribute should have.

(b) Choose the class the attribute should belong to.

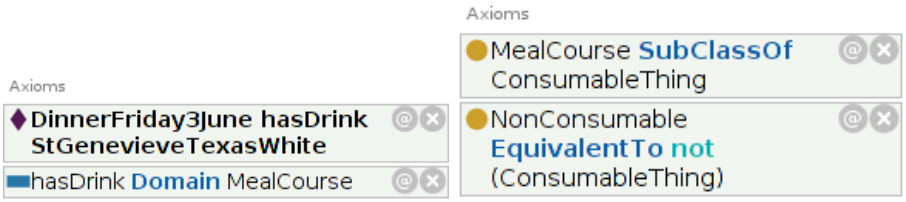
Figure 3.12: Two dialogues encountered when defining a new query attribute

In our test ontology, most DinnerCases are inferred to be members of the *MealCourse* class as well as *DinnerCase*. In Figure 3.13(a) we see the explanation provided by Protégé as to why this is the case. We see that the dinner case is a member of DinnerCourse because it has the property *hasDrink*, which is specified to have the domain of MealCourse.

It has also been inferred that MealCourse is disjoint with the class *NonConsumable*. The explanation provided is that MealCourse is a subclass of ConsumableThing, and NonConsumable is equivalent to the complement of this class. This explanation is shown in Figure 3.13(b).

As we see from the explanations in Figure 3.13 they would not be of very much use for a novice user. The explanations are simply a trace of the axioms that result in the inferred statement that we have requested an explanation for. It is of the same quality as the explanation provided by the *Inference web* system, shown in Figure 2.1. They may, if we have understood the Inference Web system correctly, slightly more helpful. In seems like in the Inference Web system it is not possible to click on any of the elements comprising the explanation, this is however possible in Protégé.

We do however doubt that the intent of these explanations is to explain things



(a) The reason a dinner is inferred to belong to the *MealCourse* class.

(b) The reason a dinner is inferred to be disjoint with *NonConsumable*.

Figure 3.13: Protégé offers explanations for what has been inferred.

to novice users. It is more likely that they are intended for debugging, and perhaps understanding, the active ontology, and it is not very likely that persons with these goals are novices.

3.4.8 Similarity explanation

The ability to explain the similarities between query and instance comes mainly from the improvements done to myCBR 3 which we described in section 3.3. We think it is useful for the user to be able to see a decomposition of the similarity computation, such that he better can understand how the attributes he specifies in the query affects the result. Figure 3.14 shows the result after executing a query

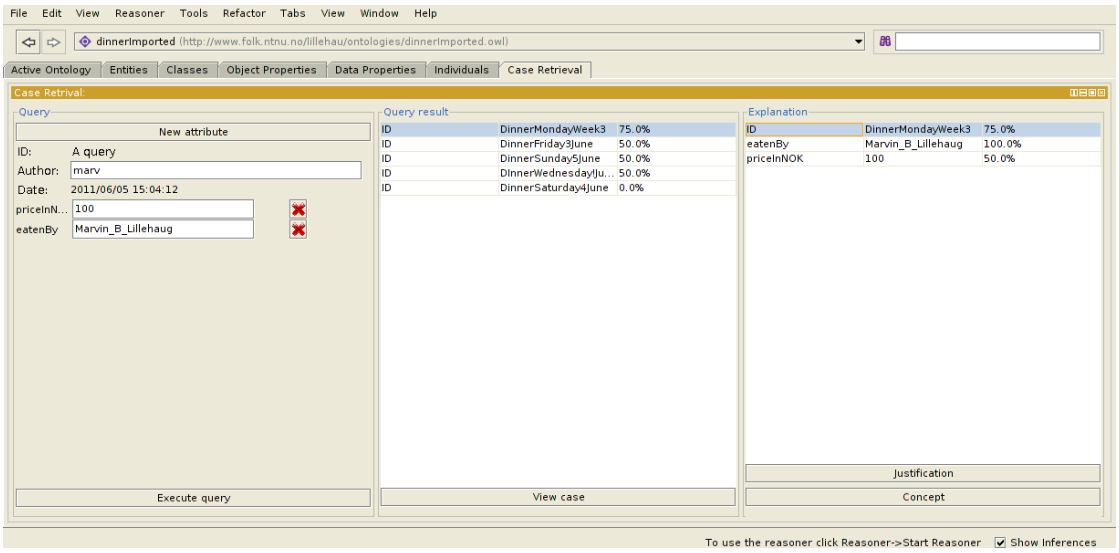


Figure 3.14: When retrieved cases are selected, its similarities are shown.

with the integer attribute *priceInNOK* having concept *Cost* and instance attribute *eatenBy* having concept *Person*. The query results show five cases ordered by decreasing similarity value.

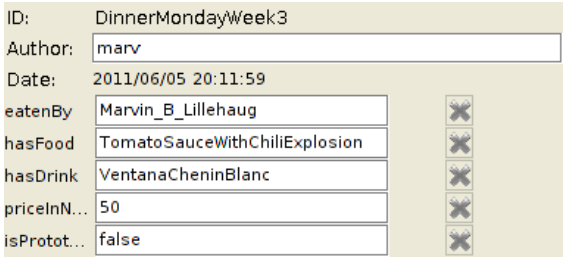


Figure 3.15: View of a single case

By pressing *View case* the selected case is opened in a separate window, listing all its attributes, as shown in Figure 3.15. When a case is selected its similarity explanation is shown in a separate panel on the screen, the rightmost panel in Figure 3.14. The panel shows in the top row the same information as in the query result, the case name and the total similarity. In the following rows each attribute specified in the query is shown with the similarity value between the corresponding attribute in the selected case instance. We see from the figure that the total similarity was 75%, with sub similarities of 100% and 50% for eatenBy and priceInNOK respectively. Having selected the first row in this panel and pressing *justification* the window shown in Figure 3.16 is shown. This shows a slightly more detailed view of the instance the query has been compared to. As in the view shown in Figure 3.14 only the attributes present in the query is shown. When the query has defined an attribute that is not present in the case instance the similarity values shown is *N/A*.

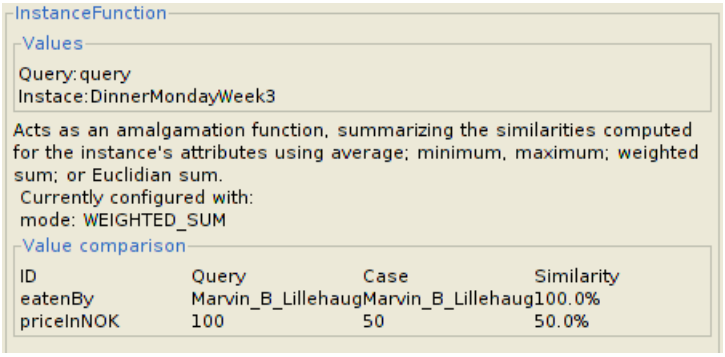


Figure 3.16: Justification explanation of how the Instance function works.

In addition to show the query and case instance attribute values with their similarity in Figure 3.16, an explanation of how the similarity metric works is shown. We can see that the *InstanceFunction* is a function that simply computes an aggregate of the similarity values of the instance's attributes using a configured method. As we can see that the shown function is configured to compute a weighted sum of the similarities when compared to a query. We have chosen not

to implement the possibility to adjust the weights for the attributes, so all weights are 1.0. If it was possible to adjust the weights it would be natural that their value was shown in this window.

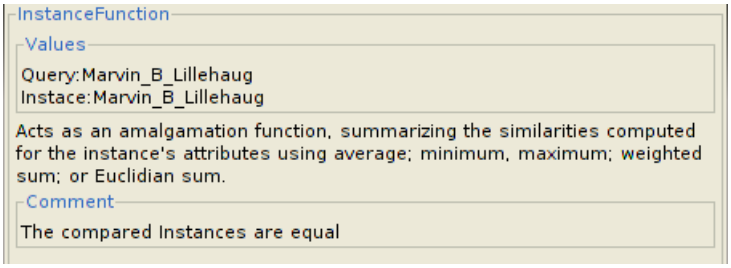


Figure 3.17: Explanation of why the eatenBy attribute has 100% similarity.

Many attributes are, as stated in section 3.4.4, natural to model as instances that may or may not have attributes of their own. The query attribute *eatenBy* is such an attribute, representing one of the persons that has attended a dinner.

In figure 3.17 we see the explanation for why this attribute got a similarity value of 100%, the person instance specified in the query is the same person as specified in the case instance. Here we see that the InstanceFunction has been implemented such that it, instead of comparing all attributes, just concludes that the instances are equal when they have the same name. This is valid as long as the name equals the fragment of the instance's URI. The fragment is found after a "#" at the end of the main URI. The full URI may be www.ont.com/ontology#name the fragment is then name.

When pressing *justify* on the attribute priceInNOK we get what is shown in Figure 3.18, showing the results of similarity measurement with the *IntegerFunction*. We see that this function has a large number of adjustable parameters that affect the resulting similarity value.

3.4.9 ConceptExplanation

myEACBR provides explanation for the concept the selected attribute belongs to when pressing *Concept*. For the two attributes in our query this is `Person` and `Cost` for eatenBy and priceInNOK respectively. The concept explanations originate from both online and offline sources, providing several explanations for each concept given that each source contains information about it. We have implemented four knowledge sources for concepts that we now will describe.

Wordnet

Wordnet is a lexical database containing more than 118,000 different word forms and more than 90,000 different word senses, semantic relations between words[Miller,

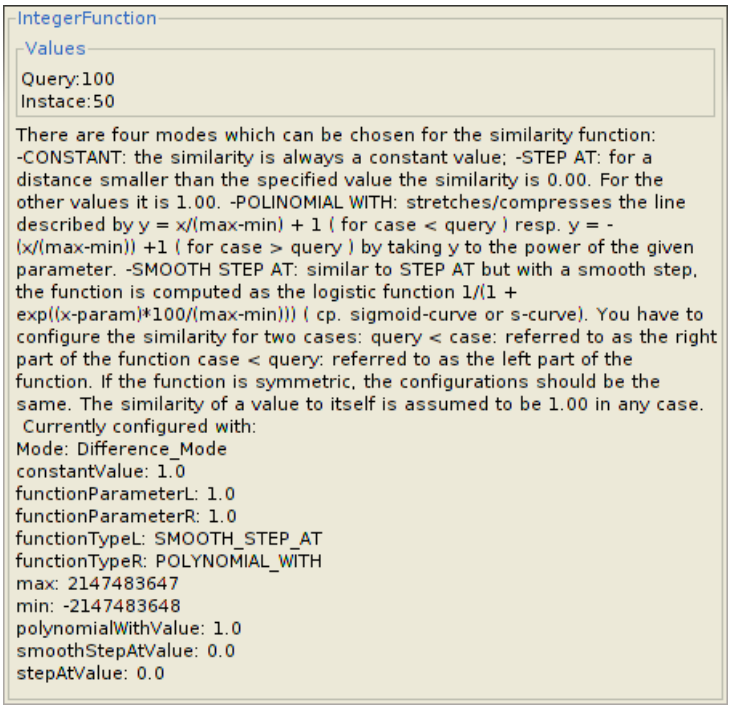


Figure 3.18: Explanation of how the IntegerFunction works and the resulting similarity for the priceInNOK attribute.

1995]. We do however not make use of all these relations, only the description and synonyms of the words being names of the concepts asked to explain.

When the user demands a concept explanation the Wordnet *dictionary* is queried, resulting in zero or more word definitions. When the result contains more than one word, the user is given the ability to choose which meaning of the word should be used, as shown in Figure 3.19. Both the definition of the meaning of the word and its synonyms are shown in this dialogue. Since there are other relations between word stored in Wordnet it would be possible to display more information about each word, but this was not prioritized as high as other matters in the project.

We have used an offline version of Wordnet found at `http://wordnet.princeton.edu/wordnet/download`, as well as the Java library *JWI* available at `http://projects.csail.mit.edu/jwi`.

Wikipedia and Wiktionary

`wikipedia.org` is an online encyclopedia written collaboratively by volunteers. It is in most cases considered to be as good as traditional encyclopedias, especially regarding science subjects[Giles, 2005]. We assume that the reader is familiar with

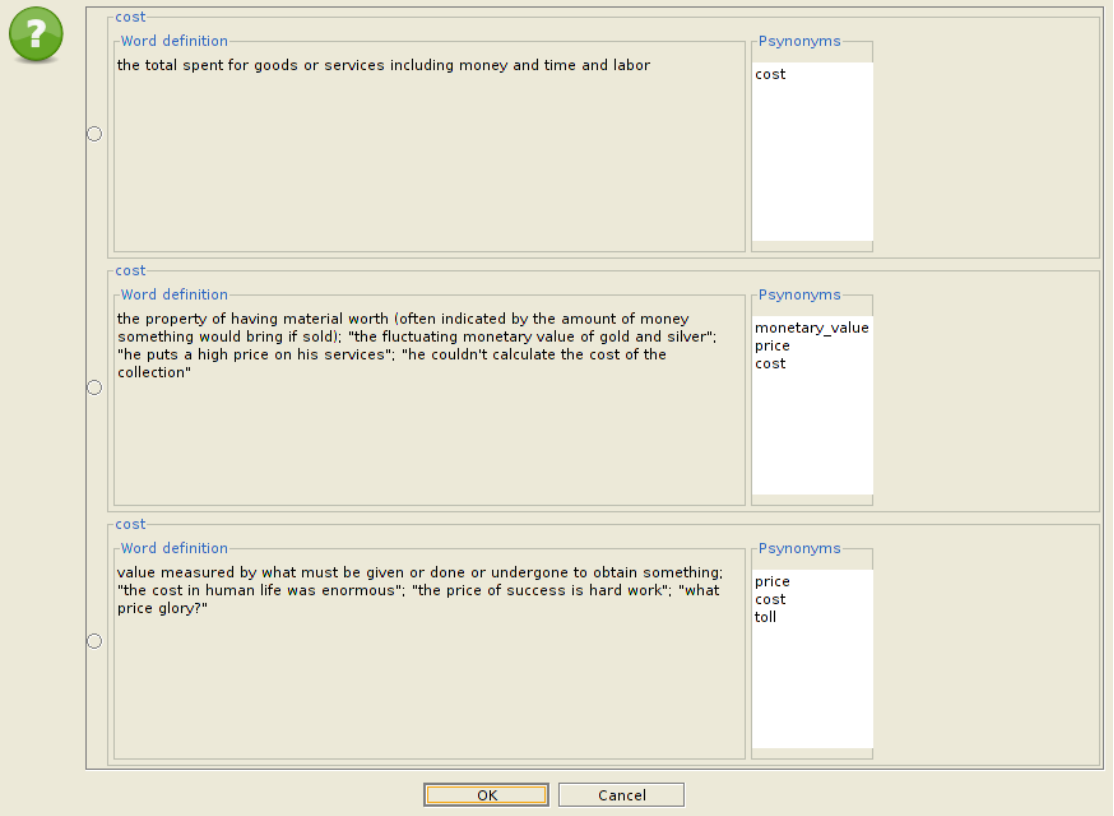


Figure 3.19: Concept explanation of cost from wordnet.

Wikipedia, and will not go into further details.

`wiktionary.org` is the dictionary equivalent to Wikipedia, containing information about words. It is, like Wikipedia, written collaboratively by volunteers and most entries include more information than a normal dictionary, such as additional information typically found in thesauri and lexicons. We consider both these sources reliable enough for our purpose.

Both pages exists in several languages, but we have made use of the English versions exclusively.

The explanations we construct from these two sources are simply the text that is available on the page with the same name as the concept, as shown in Figure 3.20. This could be enhanced by providing some option to the user to change to a different page when there are alternative pages with topics sharing a name, but the API provided[9] did not have such functionality "out of the box" so this feature would have been quite complex to implement.

[9] `http://en.wikipedia.org/w/api.php`

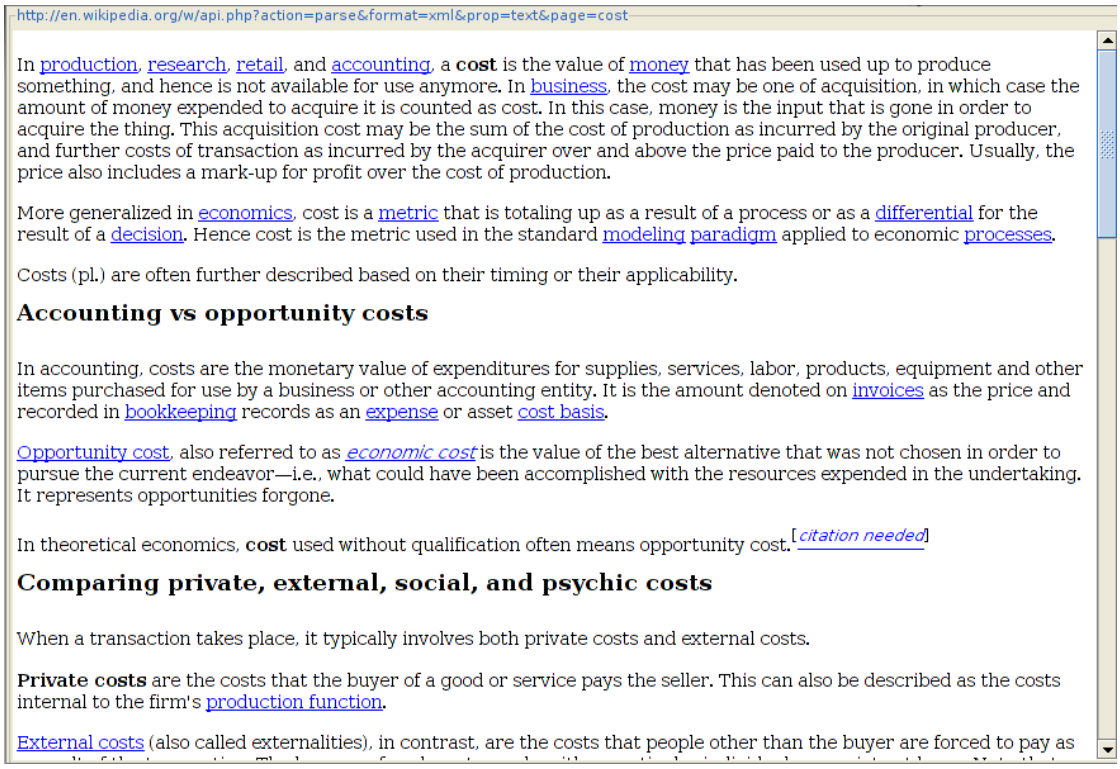


Figure 3.20: Concept explanation originating from a wiki knowledge source.

Wolfram Alpha

The last concept explanation source we have implemented is *Wolfram Alpha*(`http://www.wolframalpha.com`). Wolfram Alpha is a "computational knowledge engine", matching queries to a controlled library and computing answers and relevant visualizations from a core knowledge base of curated, structured data[Hoy, 2010]. The results returned when queried contains the meaning which the engine assumes was intended by the query as well as a number of "pods", each containing some bit of information about the subject. As we see in Figure 3.21, where the results of the query *cost* is shown, many types of information is available.

An API to query Wolfram Alpha by other means than through a browser is available[10] and is free, but the number of queries available is limited to 2000 per month. It provides the data in several formats and provides a easy way to navigate to alternative interpretations of the query in our plugin.

[10] `http://products.wolframalpha.com/api`

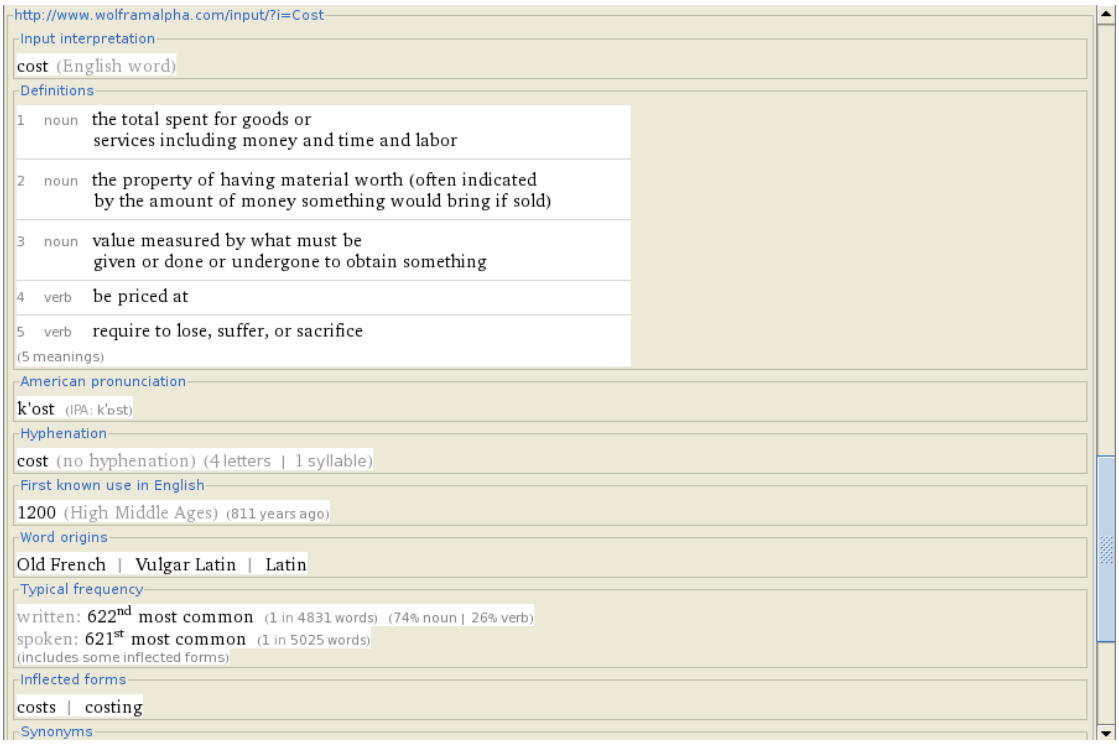


Figure 3.21: Concept explanation of cost originating in Wolfram Alpha.

3.4.10 Explanation provenance

To trust an explanation it is important to be aware of its *provenance*, where it came from. This is important for both the experts behind the system as well as the users of it. When the experts need to verify the explanations given, the system needs to be able to tell how the answers where derived such that the experts know that it was not based on a bug or some other anomaly. The same reason applies for the users of the system, they need to know that the knowledge used originates from a trusted knowledge source and that valid methods are used for computing answers from these sources.

To some degree this is the same as the *transparency* and *justification* explanation, showing the steps gone through when computing the answer and why this method was used.

In our plugin and myCBR 3 it is possible to examine the whole tree of similarities resulting in the final similarity value, and the similarity metrics underlying the computation is described in the justification explanations.

In addition to this which source giving each particular concept explanation is shown, as well as a link showing where to obtain a similar explanation of the concept in a browser. Examples of this are seen in the top of Figures 3.20 and 3.21.

In the case of Wikipedia, Wiktionary and Wolfram Alpha these links will show exactly the same explanation as shown in our plugin. In the case of Wordnet, in which we used an offline method, we where not able to find a method for specifying which of the different meanings of a word wanted, so the link provided results in a page containing the same information as shown in Figure 3.19.

A comment we can make on the three former knowledge sources is that they themselves have quite good provenance. Wolfram Alpha does not list specific references for the different items in their result, but provide information about the sources for each topic of knowledge as a link ("Source information"). For all Wikipedia entries it is possible to view the history of everyone that has contributed to the entry and what they have contributed. We will not go into the discussion of how reliable this information is since this is far outside the scope of the project.

3.4.11 Saving for later use

When the user has chosen the desired interpretation of the concept to be explained he is shown an overview where the explanations from every knowledge source is listed. He then has the possibility to save the explanation, such that the chosen explanations are used next time an explanation of the concept is requested. If this was a "real-life" application it would perhaps not be so likely that the user would know which of the descriptions of the concept that was the correct one, so this feature is perhaps more likely to be used by the knowledge engineer.

When explanations are saved in the ontology, instances representing each of the explanations from the different sources are saved. These instances have data properties containing the textual explanation, concept name, the name of the class that generated the explanation and a link to its representation online. The purpose of this is, as said, so that the particular explanation is used next time a user requests the concept explained. It could also be used such that online access is not needed to generate the explanation, but the Wolfram Alpha explainer will not give the exact same explanation in this case.

The reason the explanation will differ is because the explanation, as shown in Figure 3.21 is based on images generated by the server. The same explanations are available in text form, and we do fetch these as well, but they naturally do not have the same quality and explanatory power in plain text.

3.4.12 OWL integration

The OWL API[11], which we already have mentioned a few times, is the reference implementation for creating, manipulating and serializing OWL Ontologies. The

[11] `http://owlapi.sourceforge.net`

latest version of the API is focused towards OWL2.

Since the web page for the API contains both the source code and examples of usage it is fairly easy getting started manipulating the ontology and extracting the desired information from it. And since it is this same API that is used in Protégé, there was no problem using the ontology that was already loaded and do as we pleased with it. We did however not find a way such that when changes were made, either by us or the reasoner we controlled, Protégé would load these changes seamlessly. This means that when we did changes in the ontology, Protégé shows a notification warning that the ontology has changed and asking if it should reload the ontology. Also, even though we have obtained the reasoner through Protégé's reasoner manager, it did not show any signs that inferencing had been done in the other tabs in Protégé. This is not a problem, but it was surprising that this was the case.

There are two classes that are responsible for the integration towards OWL, which we will now describe briefly.

OWLToMycbr

This class is responsible for converting the concepts and instances represented in the ontology into in-memory representations within myCBR. It simply starts from the concept *Thing*, recursively traversing all sub concepts, creating concept objects, attributes and instances.

Each data property value has a specific type, e.g. boolean, and a corresponding myCBR object is created for the particular attribute type. A boolean data property naturally is represented as `BooleanAttribute`.

Since the ontology is traversed in this manner, even the CBR cases are represented in the myCBR concept hierarchy. This means that all that, in order to find all cases one simply has to access the concept with the name *CBR-Case* and grab all its instances.

ConceptExplanationIntegrator

As the name indicates, this class integrates concept explanations into the ontology. For each concept explanation it creates an instance under the concept *Concept-Explanation* and adds properties to it. It adds properties for concept name, links, textual representation of the explanation and the class that created the explanation.

The class also fetches stored explanations from the ontology and creates ConceptExplanation objects from them, reusing them as described in section 3.4.11.

Chapter 4

Evaluation

In order to justfiy the results of any project, scientific; product development; or any other kind of project, it needs to be evaluated. How this evaluation is conducted naturally depends on the nature of the project and how much attention has been given to the fact that an evaluation is going to be performed. A natural way to evaluate the development of a product is to measure the quality of the product as well as how it has been recieved in the marked.

4.1 Guidelines for AI research

In [Cohen and Howe, 1988] the authors present a model for AI research and describe guidelines that are appropriate for each of the stages in the model. The authors argue that evaluation provides a basis for the accumulation of knowledge. If the work is not evaluated it is not possible to replicate the results. Because of this, the basis for accepting the results, as well as the ideas and assumptions underlying the results are weaker.

The model presented consists of five steps. Refine a topic to a task and a `view` of how to accomplish the task; refine the view into a specific method; implement the method; design experiments to test the implementation; and run the experiments. The authors point out that this a bit idealized, but that few modifications to `standard` AI practice is required. And though is seems very `top down`, the criteria presented are beneficial to keep in mind regardless of the type of project at hand. We will now summarize the five stage model presented by the authors.

4.1.1 Refine a topic to a task

The first stage presented is to refine the topic that has been found interesting into what the authors call `a task` and `a view`, defined as something we want the

- Is the task significant?
- Is your research likely to meaningfully contribute to the problem?
- As the task becomes specifically defined for your research, is it still representative of a class of tasks?
- Have any interesting aspects been abstracted away or simplified?
- What are the subgoals of the research?
- How do you know when you have successfully demonstrated a solution to the task?

Figure 4.1: Criteria for evaluating research problems

computer to do and a rough idea about how to do it respectively.

To guide this process, the questions listed in Figure 4.1 has been formulated. The answers to these questions listed justify that the project initiated is a viable one. When stating the significance of the task it should be made clear whether the problem previously has been defined, and if not, how the current approach is an improvement over the previous approaches. After the task has been formulated it has to be clear that it is tractable and how a successful solution is recognized. If the problem is not well enough understood, it may be that a successful solution is easily recognized. It should also be clear what aspects have been abstracted out or simplified, both in the previous definitions as well as the current one. If the current definition is based on previous ones, it may be that the results end up being invalid if previous abstractions and simplifications are not considered.

4.1.2 Design the method

At the second stage it should be clear exactly what the task comprises, including how and why it has been restricted in this particular way. The second stage naturally relies on the first stage and it define exactly what the method for obtaining results concerning the well defined problem task is. When designing the method it is important to state how it differs from existing methods and technologies, this is ensured by posing the questions in Figure 4.2. This may include how the situations accounted for, the computational complexity or other significant factors differ. If it depends on other methods these have to be described, and both our assumptions and the assumptions of these methods have to be accounted for. It should also be clear under which circumstances the method will work, and also why it works.

- How is the method an improvement over existing technologies?
- Does a recognized metric exist for evaluating the performance of your method (for example, is it normative, cognitively valid)?
- Does it rely on other methods?
- What are the underlying assumptions?
- What is the scope of the method?
- When it cannot provide a good solution, does it do nothing or does it provide bad solutions or does it provide the best solution given the available resources?
- How well is the method understood?
- What is the relationship between the problem and the method?

Figure 4.2: Criteria for evaluating the method

4.1.3 Implement

- How demonstrative is the program?
- Is it specially tuned for a particular example?
- How well does the program implement the method?
- Is the program's performance predictable?

Figure 4.3: Criteria for evaluating the implementation

The third stage involves implementing the designed method and program, questions to have in mind in this stage is shown in Figure 4.3. Since it is not usual to publish the program itself, it is critical to describe the program thoroughly. This includes describing how well it is possible to evaluate the program's internal and external behaviour and what its limitations are. If the implementation process has uncovered any unforeseen problems or aspects these should be explained, as well as the range of examples and test cases that have been used to verify the implementation of the method. These things should also be discussed in more detail in the fourth stage.

- How many examples can be demonstrated?
- Should the program's performance be compared to a standard such as another program, or experts and novices, or its own tuned performance?
- What are the criteria for good performance? Who defines the criteria?
- Does the program purport to be general (domain-independent)?
- Is a series of related programs being evaluated?

Figure 4.4: Criteria for evaluating the experiments' design

4.1.4 Design experiments

When the method has been fully implemented, experiments have to be performed in order to document how well the particular method perform. Although this stage comes after the implementation stage it is important that experiments have been kept in mind during the implementation. If the process of executing the experiments and obtaining accurate results are cumbersome, it is likely that they have an unnecessary low quality. To avoid this the questions in Figure 4.4 have been formulated.

When designing and describing the experiments it is important to specify which measures are used to do the evaluation. In some cases there is a standard to compare to, in others there may be other programs or a panel of experts and novices. The underlying assumptions, simplifications and generalizations have to be kept in mind when designing experiments. The validity of the results depend on that the same conditions are present both when our method is run and when the cases we are comparing with were ran. If this is not possible the differences in running conditions should be described and their impact discussed.

The authors present six types of studies that can be used when evaluating.

Comparison A set of measures are selected, by which both the program and standard/other programs are compared.

Direct assessment It is not possible to strictly define measures by which to compare, so an expert has to assess the performance of the program.[1]

Ablation and substitution Components of the program are added and/or removed in order to determine and analyze how the different parts contribute to the final solution.

[1]This can be seen as a subtype of comparison studies

Tuning The system is tuned to perform well on a set of test data, then it is tested with other data.

Limitation The program is tested at its known limits by for instance providing noisy data.

Inductive The program is tested against a general or new and not before seen problem in order to show that it is general.

4.1.5 Evaluate the results

- How did program performance compare to its selected standard (for example, other programs, people, normative behavior)?
- Is the program's performance different from predictions of how the method should perform?
- How efficient is the program in terms of space and knowledge requirements?
- Did the program demonstrate good performance?
- Did you learn what you wanted from the program and experiments?
- Is it easy for the intended users to understand?
- Can you define the program's performance limitations?
- Do you understand why the program works or doesn't work?

Figure 4.5: Criteria for evaluating the results

The authors claim that the activities performed in the last step, evaluating the results, is what most people regard as evaluation. It is then easy to forget that the evaluation is critical and it gets less focus than necessary in the early stages. The questions posed in the fifth stage are shown in Figure 4.5.

In addition to presenting the program's performance compared to the selected comparison measure, it should also be clear of much resources it uses. If it is not clear why the selected approach work, or does not work, this should be discussed and it should be stated whether the desired knowledge has been obtained.

4.2 Evaluation method

When we started this project we had little knowledge about the role of explanations in computer systems and especially in CBR systems. And because of this one of

the main goals of this project was to gain knowledge about explanations with a focus on their role and presence in CBR systems. We have tried to structure our search in the background knowledge regarding this field, but we do not know whether it is good enough. We do however not try to evaluate the background chapter.

When creating our plugin for Protégé we did have the background knowledge presented in Chapter 2 in mind and we are going to evaluate the results presented in Chapter 3 and are going to evaluate our results in the light of this, we will especially focus on the explanation goals presented in 2.1.2 as well as the criteria presented in 2.1.3. Since our project has been concentrated on gaining knowledge and experience on the subject of explanations constructed by computer systems, we have no external tests that have been run to test the program's performance in any way. Nor have we had time or resources to perform a user test to see whether the presented explanations were useful to the user. This means that the only evaluation of the plugin implemented will be a discussion of how well it fulfills the goals and criteria presented in the background material.

Even though we have presented some evaluation of code quality; common practices and such, when presenting our improvements of myCBR 3 in section 3.3, we will not preform any evaluation of the code quality in our own code. We have as long as it is practical, not just dogmatic, followed the guidelines and practices presented.

In addition to this we will naturally try to incorporate the relevant criteria presented in the previous section into our evaluation.

4.3 Evaluation

Since we have focused on generating explanations we do not have any formal tests that we have run in order to evaluate our results. We have created a system that is capable of generating two kinds of explanations, which we will evaluate in terms of the knowledge presented in Chapter 2.

The only thing we think could have been evaluated by more formal test is the criteria of *low construction overhead*. But this would require to obtain or create a substantially larger dataset, and we would probably also stumble upon the indexing problem which we know needs to be solved in order to have an efficient CBR system.

4.3.1 Similarity explanations

The similarity explanations we have presented are a combination of *transparency* and *justification* explanations. The final similarity value can always be decom-

posed into all of the individual similarity measurements comprising it, making the explanation transparent. The justification part comes from the fact that each explanation contains information about what method was used to calculate the similarity value. In particular the name of the similarity function as well as a description of how it works and what configuration parameters has been set.

Regarding the content of the similarity explanation, in particular the justification part of it, there are some elements missing. The elements we think are missing are for the most part visual and if present would make the explanations easier to understand for a novice user, increasing the ratification and confidence in the system. It would for instance be easier to understand how the `IntegerFunction` works if a graph was presented such that the user could see how the similarity values were distributed.

The explanations do however have high *relevance*, unless the description of a similarity function contains irrelevant information. Since the function description is exclusively text, there may be some problems in cases where it contain concepts that should have been linked such that they could be explained. To do this however, would require a great deal of effort.

The explanations, since they are transparent, have high *fidelity*; *verification*; and *duplication*, exposing what knowledge has been used to generate the explanation.

The *scrutability* of these explanations is low. There are no other means of affecting how the explanations are constructed than to change the code itself. Given more time and resources this could be improved by implementing the functionality found in myCBR 2, adjusting and creating similarity functions within the application.

4.3.2 Concept explanations

The concept explanations presented are not really constructed from a knowledge source within our system, but rather an aggregate of other knowledge source outside our system. We merely fetch descriptions from other sources based on the name of the concept for which an explanation has been requested. That an explanation should be constructed from a particular knowledge source is of course not a requirement or criteria. It does however mean that the explanation given is not necessarily coherent, since it comprises up to four different explanations of the concept.

A solution to this could be to make it possible to those between the explanations from the different sources, as is done “internally” when several alternatives exist within one knowledge source. This is however not a solution that would be viable in an end-user-system, as pointed out earlier the task of engineering the knowledge in the system is not the task of a user.

One could argue that what we have presented in regards to choosing between explanations provided by several knowledge sources could serve as a prototype for a knowledge engineering tool. This would however require a more sophisticated method, not just concatenating the results from the different sources.

We are indecisive to whether the transparency and justification aspects of the concept explanations should be evaluated as good or poor. The system does not show the user how the explanations were generated, it only gives it to the user along with a link indicating where the online version might exist. It is indicated to the user where the information is originating from, and that several alternatives are offered when there are several entries with the same title. Because of this, one could argue that the explanation is transparent and justified since it is fairly clear that the explanation is gathered from this knowledge source, and that this is the entries the particular source had for the concept in question.

When the user requests a concept explanation, he is presented with several explanations from different sources. Since we do not really control what these explanations contain, other than that we do know where they originated from, we do not know whether they really are relevant. There are two problems with the approach presented in this regard. When a concept has an ambiguous name it may be that the explanation received from our knowledge sources is a completely different concept. The other problem is that if the first explanation was sufficient, the latter is not really relevant to the user.

The concept explanations are however in most cases an good explanation of what the concept in question is, and how it is used. The four sources we have used are quite good and reliable.

So we conclude the evaluation of concept explanation by pointing out that if an eventual user is satisfied by the concept explanations themselves, this is not likely to be because we have massaged the knowledge obtained from other sources, merely that the entry obtained from these sources explained the concept well.

4.3.3 myCBR 3 as a framework

As indicated by the work presented in Section 3.3, there are some issues with the myCBR 3 framework. Some of the issues have been addressed by our improvements, but we still think that it has potential for improvement. The reason we chose to use it was because we wanted to utilize OWL, and Protégé are the natural choice for editing OWL documents. Since myCBR 2 seemed to work rather well for the previous version of Protégé it seemed like a good choice to use myCBR 3. As we mentioned when presenting myCBR in Section 3.2, myCBR 3 is a standalone framework separate from Protégé. As it stands right now, after we have done some improvements, myCBR 3 is an decent framework but still has some way to go before all of its components are tuned such that it is intuitive and fits

naturally together. We will come back to myCBR when discussing further work in Section 5.1.

4.3.4 Protégé

As mentioned when evaluating myCBR 3 in the previous section, Protégé seems to be the best choice for editing ontologies. It is quite intuitive to use and when one has gone through the fundamentals of OWL, as well as Rector et al. [2004] which summarizes the most common mistakes done by novel users, it is easy to create a consistent and descriptive ontology. It is also easy to check whether the ontology is consistent by simply running a reasoner.

Chapter 5

Conclusion

We have in project this worked to fulfill and answer the goals and question presented in the introductory chapter, specifically Section 1.2, which we will summarize now.

Goal 1 Find the existing solutions and methods for explanation-aware and mixed-initiative systems in the CBR and expert system literature.

Goal 2 Design and build a CBR system that is explanation-aware and are backed by OWL. I.e. Both the case base and knowledge is contained in the ontology.

Research question 1 Are there any benefits from using OWL over Frames to represent ontologies?

In Chapter 2 we presented the background material found relevant to the subject of creating *explanation-aware* and *mixed-initiative* systems and described some of the previous attempts at doing so. In the background material we have presented how to classify explanations based on their goals and kinds, and how to evaluate the explanations given. Even though we focused mainly on CBR systems we also presented some background material concerning expert systems. This was natural since work on these systems precede work on CBR systems. After presenting what the concept of *case-based reasoning* and the role of explanations in this regard, we briefly mention the notion of *mixed-initiative systems* where the interaction with the system may initiate interaction to a greater extent than in traditional systems.

To better understand how reasoning in the presented systems are performed we researched knowledge representation, which we presented next. We describe representations such as *Production rules*, *Dynamic memory*, *Semantic net*, *Frames* and *OWL*. We observed that OWL has a formal specification of its syntax and semantics, while Frames does not have this to the same degree. Because the two

knowledge representations solves the same problem we concluded that OWL seems to be the natural choice to use for representing knowledge about cases and domain knowledge. Simply because OWL has more formal specification and has a broad area of usage through *the semantic web*. This also lead to that we described OWL in greater detail.

Based on the presented background material we implemented an application performing the retrieval step in the CBR cycle as a plugin to Protégé 4.1. It was our goal that the system should use OWL as knowledge representation and that it should be explanation-aware. We have concluded that OWL is suitable for this task and that we have to some degree succeeded in making the application explanation-aware. It is able to explain concepts using external knowledge sources and is able to explain why the retrieved cases have the particular similarity value that they have. We used myCBR 3 as a framework for the CBR part of our implementation. We where however not pleased by the code quality in this framework thus decided to improve it by using software engineering best practices.

5.1 Further work

The implementation we have presented is far from something usable for an end-user. The similarity explanations may to some degree be useful, but introducing more graphical explanations with for instance graphs visualizing the similarity functions would increase the quality significantly.

In our evaluation of the concept explanations we concluded that it should not be up to the end-user to decide which of the alternative explanations received from a knowledge source should be used to explain a concept. We conclude that this would however be a useful tool for a knowledge engineer when initializing the knowledge base. It is not possible for the user to neither define weights for the attributes nor set the parameters for the similarity functions. In cases where there exists several possible similarity functions for an attribute it is not possible to select which one to use. Much of the work of specifying cases and their attributes have been done Protégé itself, since the task of specifying cases are more specialized than modifying the ontology it would be helpful to have a separate interface for this.

Since we started this project with the intention of creating a CBR system it would also be natural to implement the three other steps in the CBR cycle.

Bibliography

Agnar Aamodt. Explanation-driven case-based reasoning. *Lecture Notes in Computer Science*, 837(2):109–143, October 1994. ISSN 0269-2821. doi: 10.1007/3-540-58330-093. URL http://www.springerlink.com/content/71rx81m1401p8511/.

Agnar Aamodt and Enric Plaza. Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AI communications*, 7(1): 39–59, 1994. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.15.9093&rep=rep1&type=pdf.

David W Aha, Tucker Maney, and LA Breslow. Supporting dialogue inferencing in conversational case-based reasoning. *Advances in Case-Based Reasoning*, pages 262–273, 1998. URL http://www.springerlink.com/index/M8NRQ3VLQPR75J8J.pdf.

David W Aha, LA Breslow, and H Muñoz Avila. Conversational case-based reasoning. *Applied Intelligence*, pages 9–32, 2001. URL http://www.springerlink.com/index/J62QX5882W795268.pdf.

David W Aha, David Mcsherry, and Qiang Yang. Advances in conversational case-based reasoning. *The Knowledge Engineering Review*, 20(03):247, May 2006. ISSN 0269-8889. doi: 10.1017/S0269888906000531. URL http://www.journals.cambridge.org/abstract_S0269888906000531.

OSG Alliance. OSGi Service Platform, Core Specification, Release 4, Version 4.1. *OSGi Specification*, 2007. URL http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:OSGi+Service+Platform+Core+Specification#0.

Franz Baader, Ian Horrocks, and Ulrike Sattler. *Description Logics*, volume 101, chapter 3 Descript, pages 122–169. Elsevier Science Ltd, January 2008. ISBN 0444522115. URL http://www.comlab.ox.ac.uk/people/ian.horrocks/Publications/download/2007/BaHS07a.pdf.

Daniel Bahls and Thomas Roth-Berghofer. Explanation support for the case-based reasoning tool myCBR. *PROCEEDINGS OF THE NATIONAL*, pages 1844–1845, 2007. URL http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Explanation+Support+for+the+Case-Based+Reasoning+Tool+myCBR#0.

Sean Bechhofer, Raphael Volz, and Phillip Lord. Cooking the Semantic Web with the OWL API. *The SemanticWeb-ISWC 2003*, pages 659–675, 2003. URL http://www.springerlink.com/index/BVN0Q18LHRHB2TL4.pdf.

Gilad Bracha. Generics in the Java programming language. *Sun Microsystems, java. sun. com*, pages 1–23, 2004. URL http://www8.cs.umu.se/kurser/TDBB24/HT05/jem/download/generics-tutorial.pdf.

V.K. Chaudhri, Adam Farquhar, Richard Fikes, P.D. Karp, and J.P. Rice. Open Knowledge Base Connectivity 2.0. 3 (Proposed). *Artificial Intelligence Center of SRI International and Knowledge Systems Laboratory of Stanford University*, 1998. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.136.489&rep=rep1&type=pdf.

WJ Clancey. The epistemology of a rule-based expert system —a framework for explanation, May 1983. ISSN 00043702. URL http://linkinghub.elsevier.com/retrieve/pii/0004370283900085.

P.R. Cohen and A.E. Howe. How evaluation guides AI research: The message still counts more than the medium. *AI Magazine*, 9(4):35, 1988. ISSN 0738-4602. URL http://www.aaai.org/ojs/index.php/aimagazine/article/viewArticle/952.

Pádraig Cunningham, Dónal Doyle, and John Loughrey. An evaluation of the usefulness of case-based explanation. *Case-Based Reasoning Research*, pages 122–130, 2003. URL http://www.springerlink.com/index/dde30n32phr934ua.pdf.

Randall Davis, Howard Shrobe, and Peter Szolovits. What is a knowledge representation? *AI magazine*, 14(1):17, 1993. ISSN 0738-4602. URL http://www.aaai.org/ojs/index.php/aimagazine/article/viewArticle/1029.

Gerald Dejong and Raymond Mooney. Explanation-based learning: An alternative view. *Machine Learning*, 1(2):145–176, 1986. ISSN 0885-6125. doi: 10.1007/BF00114116. URL http://www.springerlink.com/index/10.1007/BF00114116.

Peter J Denning. Computer science : The discipline. In *Encyclopedia of Computer Science*, number July 1999, page 419. John Wiley and Sons Ltd., 2003. URL http://web.archive.org/web/20060525195404/http://www.idi.ntnu.no/emner/dif8916/denning.pdf.

B Díaz-Agudo and PA González-Calero. An architecture for knowledge intensive CBR systems. *Advances in Case-Based*, pages 37–48, 2000. URL http://www.springerlink.com/index/urjdpkjrpl03fdey.pdf.

Dónal Doyle, Pádraig Cunningham, Derek Bridge, and Y. Rahman. Explanation Oriented Retrieval. *Advances in Case-Based Reasoning*, pages 157—-168, October 2004. doi: 10.1007/s10462-005-4607-7.

a Farquhar. The Ontolingua Server: a tool for collaborative ontology construction. *International Journal of Human-Computer Studies*, 46(6):707–727, June 1997. ISSN 10715819. doi: 10.1006/ijhc.1996.0121. URL http://linkinghub.elsevier.com/retrieve/pii/S1071581996901214.

Giorgos Flouris, Dimitris Manakanatas, Haridimos Kondylakis, Dimitris Plexousakis, and Grigoris Antoniou. Ontology change: classification and survey. *The Knowledge Engineering Review*, 23(02):117–152, May 2008. ISSN 0269-8889. doi: 10.1017/S0269888908001367. URL http://www.journals.cambridge.org/abstract_S0269888908001367.

M. Fowler and K. Beck. *Refactoring: improving the design of existing code*. Addison-Wesley Professional, 1999. ISBN 978-0-201-48567-7. URL http://books.google.com/books?hl=en&lr=&id=1MsETFPD3I0C&oi=fnd&pg=PA14&dq=Refactoring:+Improving+the+Design+of+Existing+Software,&ots=pLL4t1TLa7&sig=CLR7AZx35KFSTsGOtdkfBk6oC4w.

J Gennari. The evolution of Protégé: an environment for knowledge-based systems development. *International Journal of Human-Computer Studies*, 58(1):89–123, January 2003. ISSN 10715819. doi: 10.1016/S1071-5819(02)00127-1. URL http://linkinghub.elsevier.com/retrieve/pii/S1071581902001271.

Jim Giles. Internet encyclopaedias go head to head. *Nature*, 438(7070):900–1, December 2005. ISSN 1476-4687. doi: 10.1038/438900a. URL http://www.ncbi.nlm.nih.gov/pubmed/16355180.

Anne-marit Gravem. Integrating Case-based and Bayesian Reasoning for Decision Support. Technical Report June, Faculty of Information Technology, Mathematics and Electrical Engineering at the Norwegian University of Science and Technology, 2010. URL http://daim.idi.ntnu.no/masteroppgave?id=5480.

W.E. Grosso, Henrik Eriksson, R.W. Fergerson, J.H. Gennari, S.W. Tu, and M.A. Musen. Knowledge modeling at the millennium (the design and evolution of Protégé-2000). *Knowledge Creation Diffusion Utilization*, pages 1–36, 1999. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.31.699.

W3C OWL Working Group. OWL 2 Web Ontology Language Document Overview. Technical Report October, W3C, 2009. URL http://www.w3.org/TR/2009/REC-owl2-overview-20091027/.

Mingyang Gu and Agnar Aamodt. A knowledge-intensive method for conversational CBR. *Case-Based Reasoning Research and Development*, 2005. URL http://www.springerlink.com/index/ar7k4kryj4xlc7rb.pdf.

Mingyang Gu and Agnar Aamodt. Dialog learning in conversational cbr. *Proceedings of the 19th International Florida Artificial*, pages 358–363, 2006. URL http://www.aaai.org/Papers/FLAIRS/2006/Flairs06-070.pdf.

KM Gupta. Taxonomic conversational case-based reasoning. *Case-Based Reasoning Research and Development*, pages 219–233, 2001. URL http://www.springerlink.com/index/W4AYK4C5V2D9X5PM.pdf.

M Hall, E Frank, G Holmes, and B Pfahringer. The WEKA data mining software: An update. *ACM SIGKDD*, 2009. URL http://portal.acm.org/citation.cfm?id=1656274.1656278.

Matthew B Hoy. Wolfphram—Alpha: a brief introduction. *Medical reference services quarterly*, 29(1):67–74, January 2010. ISSN 1540-9597. doi: 10.1080/02763860903485225. URL http://www.ncbi.nlm.nih.gov/pubmed/20391166.

D. Hunt, A. and Thomas. *The pragmatic programmer: from journeyman to master*. Addison-Wesley Professional, 2000. ISBN 0-201-61622-X.

AG Francis Jr and Ashwin Ram. Computational models of the utility problem and their application to a utility analysis of case-based reasoning. *To appear in the Proceedings of the Workshop on*, (Kcsl 93), 1993. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.50.9096&rep=rep1&type=pdf.

Yannis Kalfoglou and Marco Schorlemmer. Ontology mapping: the state of the art. *The Knowledge Engineering Review*, 18(1):1–31, January 2003. ISSN 02698889. doi: 10.1017/S0269888903000651. URL http://www.journals.cambridge.org/abstract_S0269888903000651.

Alfred Kobsa. User modeling: Recent work, prospects and hazards. *HUMAN FACTORS IN INFORMATION TECHNOLOGY*, 1993. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.1.9328&rep=rep1&type=pdf.

Anders Kofod-Petersen. Explanatory Capabilities in the CREEK Reasoner. *Science And Technology*, (Idi), 2008.

David B. Leake. Goal-Based Explanation Evaluation. *Cognitive Science*, 15(4): 509–545, October 1991. ISSN 03640213. doi: 10.1207/s15516709cog1504_2. URL http://doi.wiley.com/10.1207/s15516709cog1504_2.

R. MacGregor. Retrospective on LOOM. *Information Sciences Institute, University of Southern California, Tech. Rep*, 1999. URL http://www.isi.edu/isd/LOOM/papers/macgregor/Loom_Retrospective.html.

R.C. Martin. *Clean code: a handbook of agile software craftsmanship*. Prentice Hall PTR Upper Saddle River, NJ, USA, 2008. ISBN 978-0-13-235088-4. URL http://portal.acm.org/citation.cfm?id=1388398.

P. McCorduck. *Machines who think: A personal inquiry into the history and prospects of artificial intelligence*. AK Peters, Ltd., 2004. ISBN 1568812051.

DL McGuinness and P. Silva. Infrastructure for web explanations. *The SemanticWeb-ISWC 2003*, pages 113–129, 2003. URL http://www.springerlink.com/index/krty8636e7lmwdqf.pdf.

DL McGuinness, F. Van Harmelen, and Others. OWL web ontology language overview. *W3C recommendation*, 10(February):2004–03, 2004. URL http://ia.ucpel.tche.br/~lpalazzo/Aulas/TEWS/arq/OWL-Overview.pdf.

David McSherry. Strategic induction of decision trees. *Knowledge-Based Systems*, 12(5-6):269–275, October 1999. ISSN 09507051. doi: 10.1016/S0950-7051(99)00024-6. URL http://linkinghub.elsevier.com/retrieve/pii/S0950705199000246.

David McSherry. Interactive case-based reasoning in sequential diagnosis. *Applied Intelligence*, pages 65–76, 2001. URL http://www.springerlink.com/index/x7176x61pq037m52.pdf.

George a. Miller. WordNet: a lexical database for English. *Communications of the ACM*, 38(11):39–41, November 1995. ISSN 00010782. doi: 10.1145/219717.219748. URL http://portal.acm.org/citation.cfm?doid=219717.219748.

Marvin Minsky. A framework for representing knowledge. 1974. URL http://dspace.mit.edu/handle/1721.1/6089.

Tom M. Mitchell, Richard M. Keller, and Smadar T. Kedar-Cabelli. Explanation-based generalization: A unifying view. *Machine Learning*, 1(1):47–80, March 1986. ISSN 0885-6125. doi: 10.1007/BF00116250. URL http://www.springerlink.com/index/10.1007/BF00116250.

Johanna D. Moore and William R. Swartout. A reactive approach to explanation: Taking the user's feedback into account. *Natural language generation in*, pages 1504–1510, 1991. URL http://books.google.com/books?hl=en&lr=&id=6PmGC2wwiHYC&oi=fnd&pg=PA3&dq=A+Reactive+Approach+to+Explanation&ots=nprpFs7VIM&sig=mSF9Ocaoqasi4BDmeGW1FvKzscU.

M Mozina, Matej Guid, Jana Krivec, Aleksander Sadikov, and Ivan Bratko. Learning to Explain with ABML. *ailab.si*. URL http://www.ailab.si/matej/doc/Learning_to_Explain_with_ABML.pdf.

Conor Nugent, Dónal Doyle, and Pádraig Cunningham. Gaining insight through case-based explanation. *Journal of Intelligent Information Systems*, 32(3):267–295, June 2008. ISSN 0925-9902. doi: 10.1007/s10844-008-0069-0. URL http://www.springerlink.com/index/10.1007/s10844-008-0069-0.

K.O. Pedersen. Explanation Methods in Clinical Decision Support. Technical report, Faculty of Information Technology, Mathematics and Electrical Engineering at the Norwegian University of Science and Technology, 2010. URL http://daim.idi.ntnu.no/masteroppgave?id=5501.

BW. Porter, R Bareiss, and RC Holte. Concept learning and heuristic classification in weak-theory domains. *Artificial Intelligence*, 45(1-2):229–263, 1990. URL http://linkinghub.elsevier.com/retrieve/pii/000437029090041W.

Alan Rector, Nick Drummond, Matthew Horridge, Jeremy Rogers, Holger Knublauch, Robert Stevens, Hai Wang, and Chris Wroe. OWL pizzas: Practical experience of teaching OWL-DL: Common errors & common patterns. *Engineering Knowledge in the Age of the SemanticWeb*, pages 63–81, 2004. URL http://www.springerlink.com/index/PNRG2T506C2JV3MD.pdf.

H. Reichgelt and N. Shadbolt. *Knowledge Representation: an AI perspective*. Greenwood Publishing Group Inc. Westport, CT, USA, 1991. ISBN 0893915904. URL http://portal.acm.org/citation.cfm?id=574764.

Michael M Richter. The knowledge contained in similarity measures. In *Invited Talk at the First International Conference on Case-Based Reasoning, ICCBR*, volume 95, 1995. URL http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:The+knowledge+contained+in+similarity+measures#0.

Thomas Roth-Berghofer. Explanations and case-based reasoning: Foundational issues. *Advances in case-based reasoning*, pages 195–209, 2004. URL http://www.springerlink.com/index/wd88mjuhrdfyefxu.pdf.

Thomas Roth-Berghofer and Daniel Bahls. Explanation Capabilities of the Open Source Case-Based Reasoning Tool myCBR. *mycbr-project.net*, (Figure 1), 2008. URL http://mycbr-project.net/downloads/ukcbr08.pdf.

Thomas Roth-Berghofer and Jörg Cassens. Mapping Goals and Kinds of Explanations to the Knowledge Containers of Case-Based Reasoning Systems. *Case-Based Reasoning Research and Development*, pages 451–464, 2005.

S.J. Russell and P. Norvig. *Artificial intelligence: a modern approach*. Prentice hall, 2009. ISBN 0136042597.

Roger C. Schank. Language and memory. *Cognitive Science*, 4(3):243–284, September 1980. doi: 10.1016/S0364-0213(80)80004-8.

Roger C. Schank. *Dynamic Memory: A Theory of Learning in Computers and People*. Cambridge University Press New York, NY, USA, 1983. ISBN 0521248582.

Roger C. Schank and David B. Leake. Creativity and learning in a case-based explainer. *Artificial Intelligence*, 40(1-3):353–385, September 1989. ISSN 00043702. doi: 10.1016/0004-3702(89)90053-2. URL http://linkinghub.elsevier.com/retrieve/pii/0004370289900532.

EH Shortliffe, Randall Davis, SG Axline, and BG Buchanan. Computer-based consultations in clinical therapeutics: explanation and rule acquisition capabilities of the MYCIN system. *Computers and Biomedical Research*, 8(4):303–320, 1975. URL http://linkinghub.elsevier.com/retrieve/pii/0010480975900099.

Barry Smyth and MT Keane. Remembering to forget. *Proceedings of the 14th international joint*, 1995. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.17.2767&rep=rep1&type=pdf.

P. Spieker. Natürlichsprachliche Erklürungen in technischen Expertensystemen. In *Erklärung im Gespräch-Erklärung im Mensch-Maschine-Dialog on Erklärung im Gespräch-Erklärung im Mensch-Maschine-Dialog*, pages 43–57. Springer-Verlag, 1992.

William R. Swartout and Johanna D. Moore. *Second generation expert systems*, pages 543–585. Springer-Verlag New York, Inc., 1993. ISBN 0-387-56192-7.

Frode Sørmo, Jörg Cassens, and Agnar Aamodt. Explanation in Case-Based Reasoning–Perspectives and Goals. *Artificial Intelligence Review*, 24(2):109–143, October 2005. ISSN 0269-2821. doi: 10.1007/s10462-005-4607-7. URL http://www.springerlink.com/index/10.1007/s10462-005-4607-7.

Nava Tintarev and Judith Masthoff. A Survey of Explanations in Recommender Systems. *2007 IEEE 23rd International Conference on Data Engineering Workshop*, pages 801–810, April 2007. doi: 10.1109/ICDEW.2007.4401070. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4401070.

Jennifer Vendetti and Nick Drummond. Choosing between versions of Protege. URL http://protegewiki.stanford.edu/wiki/Protege4Migration.

Wolfgang Wahlster. User models in dialog systems. (Sfb 314):4–34, 1988. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.71.9428&rep=rep1&type=pdf.

Hai Wang, Natasha Noy, Alan Rector, Mark Musen, Timothy Redmond, Daniel Rubin, Samson Tu, Tania Tudorache, N. Drummond, Matthew Horridge, and Others. Frames and OWL side by side. In *Presentation Abstracts*, page 54. Citeseer, 2006. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.84.502&rep=rep1&type=pdf#page=57.

L. Richard Ye and Paul E. Johnson. The Impact of Explanation Facilities on User Acceptance of Expert Systems Advice. *MIS Quarterly*, 19(2):157, June 1995. ISSN 02767783. doi: 10.2307/249686. URL http://www.jstor.org/stable/249686?origin=crossref.

Appendices

.1 Curriculum TDT55 2010

- A. Aamodt and E. Plaza, 1994: Case-based reasoning; Foundational issues, methodological variations, and system approaches. AI Communications, 7(1):39–59.

- D. Aha, 1991: Case-based learning algorithms. Proceedings of the 1991 DARPA Case-Based Reasoning Workshop. Morgan Kaufmann.

- A. Aamodt: Knowledge-intensive case-based reasoning in Creek. (ECCBR 2004. LNAI 3155, Spinger, 2004. pgs. 1-16.)

- Padraig Cunningham: A Taxonomy of Similarity Mechanisms for Case-Based Reasoning. University College Dublin, Technical Report UCD-CSI-2008-01 January, 2008

- Janet Kolodner: Case-Based Reasoning (1993). Chapters 1 and 2.

- Janet Kolodner: Case-Based Reasoning (1993). Chapters 3 and 4.

- Z. Sun, G. Finnie, K. Weber: Abductive Case Based Reasoning. International Journal of Intelligent Systems, 20(9) pp957-983. Wiley, 2005.

- David B. Leake. : Abduction, Experience, and Goals: A Model of Everyday Abductive Explanation. The Journal of Experimental and Theoretical Artificial Intelligence, 7:407-428, 1995. 25 pages. Springer LNAI 2689.

- Belen Díaz-Agudo, P. González-Calero:. A declarative similarity framework for knowledge-intensive CBR ICCBR, LNAI Springer 2001

- Juan A. Recio-García, Belen Díaz-Agudo, P. González-Calero, Antonio Sánchez-Ruiz-Granados :. Ontology based CBR with jCOLIBRI Applications and Innovations in Intelligent Systems XIV, 2006, pp 1490-162

- K. Branting, J. Hastings, J. Lockwood: Integrating cases and models for prediction in biological systems. AI Applications 11(1):29-48 (1997)

- Cindy Marling, Edwina Rissland and Agnar Aamodt: Integrations with case-based reasoning Knowledge Engineering Review, Vol. 20, Issue 03, September 2005. Cambridge University Press. pp 241-245.

.2 Curriculum TDT70 2010

- Explanation in Case-Based Reasoning: Perspectives and Goals, Frode Sørmo, Jörg Cassens and Agnar Aamodt, Artificial Intelligence Review. Vol. 24, no. 2, pp. 109-143, 2005.

- Explanations and Case-Based Reasoning: Foundational Issues, Thomas Roth-Berghofer, Advances in Case-based Reasoning, Proceedings of the 7th European Conference on Case-based Reasoning (ECCBR 2004), pp. 389-403, 2004.

- A review of explanation methods for Bayesian networks, Carmen Lacave and Francisco Javier Díez Vegas, The Knowledge Engineering Review, Vol. 17, no. 2, pp. 107-127, 2002.

- A review of explanation methods for heuristic expert systems, Carmen Lacave and Francisco Javier Díez Vegas, The Knowledge Engineering Review, Vol. 19, no. 2, pp. 133-146, 2004.

- Infrastructure for Web Explanations, Deborah L. McGuinness, Paulo Pinheiro da Silva, The SemanticWeb, Proceedings of the 2nd International Semantic Web Conference (ISWC 2003), pp. 113-129, 2003.

- Goal-based Explanation Evaluation, David Leake, Cognitive Science, Vol. 15, no. 4, pp. 509-545, 1991

- Introspective Self-Explanation in Analytical Agents, Anita Raja, Ashok Goel, Proceedings of IJCAI-2009 workshop on Explanation-Aware Computing, Pasadena, California, July 11-12, 2009

- A Review of Explanation in Case Based Reasoning, Dónal Doyle, Alexey Tsymbal, Pádraig Cunningham, Computer Science Technical Report, TCD-CS-2003-41, Department of Computer Science, Trinity College Dublin, 2003

- Making Sense of Sensemaking 1: Alternative Perspectives, Gary Klein, Brian Moon, Robert R. Hoffman, IEEE Intelligent Systems, july/august 2006

- Making Sense of Sensemaking 2: A Macrocognitive Model, Gary Klein, Brian Moon, Robert R. Hoffman, IEEE Intelligent Systems, july/august 2006

- Learning to Explain with ABML, Martin Mozina, Matej Guid, Jana Krivec, Aleksander Sadikov, Ivan Bratko, Proceedings of the 2010 ExaCt workshop on Explanation-aware Computing

- An Evaluation of the Usefulness of Case-Based Explanation, Pádraig Cunningham, Dónal Doyle and John Loughrey, Case-Based Reasoning Research and Development
- Designing Explanation Aware Systems: The Quest for Explanation Patterns, Jörg Cassens, Anders Kofod-Petersen, The AAAI 2007 Workshop on Explanation-aware Computing (ExaCt 2007)