Using Hierarchical Knowledge Structures to implement Dynamic FAQ Systems

David Camacho[†], Maria Dolores Rodriguez-Moreno[‡], Alberto López[†],César Castro[†]

[†] Universidad Carlos III de Madrid, Departamento de Informática. Madrid, Spain.
email: dcamacho@ia.uc3m.es

[‡] Universidad de Alcalá de Henares, Departamento de Automática. Madrid, Spain.
email: mdolores@aut.uah.es

Abstract. This paper presents a knowledge representation that allows building hierarchical structures that can be used to build dynamically solutions to solve a particular user question. The goal of this work is to show how it is possible to define (and handle) a new knowledge representation that allows the integration of Case-based Knowledge into a graph-based representation that can be easily learned and managed. The combination of both, Case-based Knowledge and graphs allows to implement a flexible hierarchical structures (or *learning graphs*) that have been applied to implement a new kind of Frequently Asked Questions Systems. In these systems the output is dynamically built from the user query, using as basis structures the knowledge retrieved from a Case Base. The paper shows how the management of these cases allow enriching the knowledge base. Finally, the paper shows a specific application of this technique, in particular a Web system named DynJAQ (Dynamic Java Asked Questions). DynJAQ is a FAQ system that is able to generate dynamically several HTML guides that can be used to answer any possible question about a particular programming language (Java).

Keywords: Knowledge Representation, Case-Based Reasoning, Intelligent Web Systems.

1 Introduction

Knowledge representation and management performs crutial aspects in most of the software applications. Both topics become specially important when Artificial Intelligence techniques are used. Several features such as flexibility, adaptability, reasoning, or learning are directly related to the knowledge representation used in those systems [7].

The aim of this work is to define a knowledge representation that must be able to integrate predefined knowledge about a particular topic (cases) into a common solution. These solutions will be built using this knowledge as the basis (or atomic) knowledge structures. The interaction with the system (when a problem,

or question is proposed) will generate different solutions using several parameters provided by the users. Therefore the solution will be dynamically adapted to the user skills. To show our knowledge-based approach, the next problem will be addressed: when any student needs to learn concepts about how to program in a particular language usually some books, manuals or distribution lists are used to solve his/her problems. When these problems are very usual it is possible to build a Frequently Asked Questions (FAQ) repository to answer these questions.

The Knowledge Based approach presented in this paper allows to obtain a solution *adapted* to the user skills, using stored knowledge. We have instantiated our knowledge representation into a particular Web system. The implemented system (named DynJAQ) allows to ask questions such as: *"I want to know how to define classes, and define variables and constants, in Java"*, and gives to the user one or several solutions for this particular question.

The paper is structured as follows. Section 2 shows the Related Work. Section 3 describes the hierarchical knowledge representation proposed. Section 4 shows a Dynamic Web FAQ System, named DynJAQ, that has been implemented using the previous knowledge representation. Finally, Section 5 summarizes the conclusions of this work.

2 Related Work

This section addresses some related systems to our approach. We briefly describe the main features of those systems in areas such as: FAQ Web systems, Questions Answering systems, and Case-Based Reasoning systems.

2.1 FAQ Web systems

Usually a FAQ works in the following way: any user can consult a pre-existing list of questions with their answers in order to found a similar question that can answer his/her own problem. The user is the responsible of both, to analyse all the items in the FAQ (usually searching through previous questions asked by other users), find the most similar solutions and "reuse" or "adapt" to his/her problem. There is a huge number of Web sites that provides FAQ repositories, some interesting examples are:

- The Dynamic FAQ Database (http://products.dynamicwebdevelopers.com) is a Web Application that allows to Construct a Frequently Asked Questions Database for a particular Website. Therefore, the users may search in the FAQ Database, as well as submit questions for support. This application simply builds a database query access using a simple Web interface.
- The Hope Resource Center Dynamic FAQ (http://www.stmarys.org/ cancer/ faq/ default.asp). This FAQ provides a dynamic access through the selection of a particular issue in a list. Once any item is selected some static FAQ documents are shown.

- Web sites like *The Collection of Computer Science Bibliographies* (http:// liinwww.ira.uka.de/ bibliography/ index.html), is a traditional static FAQ based on a set of questions and their related solutions.
- Other Web sites like The Internet FAQs Archives (http://www.faqs.org/faqs/) allow to find different FAQ documents from repositories using a set of keywords.

However, developing a simple FAQ repository has several problems that could be summarized in:

1. Not all the users have the same knowledge about a particular topic. Therefore, if only a static FAQ is implemented, the questions could be over specific and the user could not understand correctly the solution proposed, or very general so the user does not find what s/he is really looking for.
2. If a Web system is implemented using only a static repository of solutions, this repository will grow up quickly and the number of documents retrieved could be very large. Therefore, the users could not find the information, or it could be very hard.
3. None FAQ system takes into account the user features, or skills, about a particular topic.
4. These kind of systems are not flexible because they store a set of predefined problems and their related solutions.

2.2 Question Answering systems

There is an important research work related to the *Question answering* (QA) systems [5]. A question answering system provides directly an answer to the user question by accessing and consulting its Knowledge Base. These type of systems are related to the research in Natural Language Processing (NLP) [6].

In recent years, and due to the evolution of the Web, has originated a new interest in the application of QA systems to the Web. In [9] is defined what is required to implement a web-based QA system. Any QA system based on a document collection typically has three main components. The first is a retrieval engine that sits on top of the document collection and handles retrieval requests. In the context of the web, this is a search engine that indexes web pages. The second is a query formulation mechanism that translates natural-language questions into queries for the Information Retrieval engine in order to retrieve relevant documents from the collection, i.e., documents that can potentially answer the question. The third component, answer extraction, analyzes these documents and extracts answers from them.

Perhaps the most popular information retrieval system, based on NLP techniques is FAQ Finder [3–5]. FAQ Finder (http://faqfinder.ics.uci.edu/) is a system that retrieves answers to natural language questions from USENET FAQ files. The system integrates symbolic knowledge and statistical data in doing its question-matching. Part of the challenge was to pre-compile as much of the system knowledge, therefore the answers could be found fast enough to satisfy

the constraints of the Web use. One issue raised by this research is the need to have the system correctly identify that a question cannot be answered.

However, the difficulty of NLP based systems has limited the scope of question answering systems to *domain specific* systems. In our approach this problem is relaxed using a general graph-based search technique integrated with a domain dependent Case-based Knowledge.

2.3 Case-Based Reasoning systems

Case-based reasoning (CBR) [1, 2, 8] solves new problems by adapting previously successful solutions to similar problems. This problem solving technique does not require an explicit domain model, so elicitation becomes a task of gathering case histories. The implementation is reduced to identifying significant features that describe a case. This case is then stored and managed by means of database techniques, and CBR systems can learn by acquiring new knowledge as new cases.

Figure 1 shows the processes involved in CBR. This general CBR cycle may be described by the following four processes: [1]

- RETRIEVE the most similar case, or cases.
- REUSE the information and knowledge stored in the case, or cases, that solve the problem.
- REVISE the proposed solution (if necessary).
- RETAIN the new solution as a part of a new case.

Fig. 1. The CBR Cycle.

A new problem is solved by retrieving one or more previously experienced cases, reusing the case in one way or another, revising the solution based on reusing a previous case, and retaining the new experience by incorporating it into the existing Knowledge Base (Case Base).

This cycle rarely occurs without human intervention. For example, many CBR tools act primarily as a case retrieval and reuse systems. Case revision (i.e.,

adaptation) is often being undertaken by managers of the case base. However, it should not be viewed as weakness of the CBR that it encourages human collaboration in decision support. Our approach allows implementing a new way to manage the cases stored in the Case Base through the use of two different kind of cases, atomic and complex (see the following Section).

3 Hierarchical Learning Graphs

This section describes how to combine a graph-based representation and case-based knowledge into a hierarchical representation that can be used to build new flexible and adaptive QA systems.

3.1 Integrating Case-based Knowledge into a Graph Structure

The proposed knowledge structure uses a graph representation as the main knowledge integration structure. The combination of graph structures and CBR knowledge implements the concept of the *learning graph*. Any learning graph can be characterized as follows:

- The *nodes* in the graph are cases extracted from the case base. Those nodes can be: *atomic*, if they only represent simple concepts, or *complex*, if they are implemented using other learning graphs.
- Any node (case) in the graph uses pre-connectors and post-connectors to represent what concepts are necessary to understand the knowledge stored in the node (pre-connectors), and what concepts are acquired if this node is learned by the user (post-connectors). Those connectors are used to define the *transitions* between the nodes in the graph.

Therefore, any learning graph is a hierarchical nested structure, where each node can be decomposed into several learning graphs, and where the leafs of this structure are implemented by simple knowledge structures (or atomic cases). Figure 2 shows a schematic representation of this structure.

To build an initial learning graph, it is necessary to define how to obtain the initial and final nodes in the graph, and how to generate from those nodes the rest of the graph. In our approach, the initial and final nodes are obtained from the user interaction. Once those nodes are defined a simple searching algorithm is used to build the learning graph. The algorithm can be summarized as follows:

1. Using the user query (and other user characteristics), the initial and final nodes are defined (see Section 4).
2. While it does not exist a path from the initial node to all the final nodes do:
   - 2.1 Retrieve from the Case Base all the cases that match with all the post-connectors of the initial node.
   - 2.2 From the retrieved cases select those whose pre-connectors connect completely all the post-connectors in the previous node.

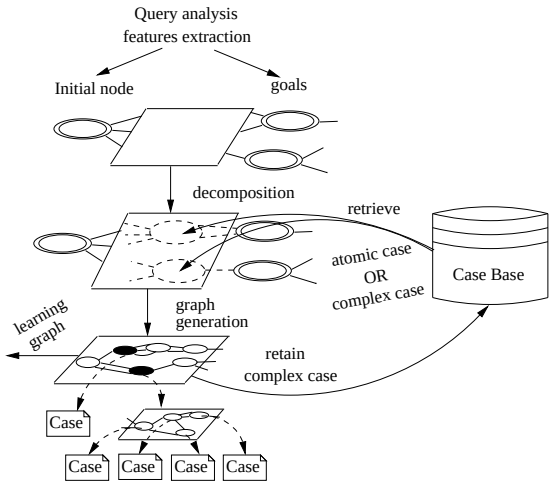

Fig. 2. Learning graph representation.

- 2.2.1 If there are more than one case that could be used to connect a particular node, then generate one learning graph for each node.
- 2.2.2 Insert those nodes in the graph.

– 2.3 If does not exist any case that match with all the post-connectors for a particular node, retrieve a set of nodes that partially match with those connectors, until all the post-connectors are connected.

- 2.3.1 For each selected node verify that all of its pre-connectors are connected, if don't, look for other nodes (cases) that can be used to join this node with the previous node.
- 2.3.2 Insert those nodes in the graph.

3. For each learning graph generated, look for a path (solution) from the initial node to the final node (or nodes).

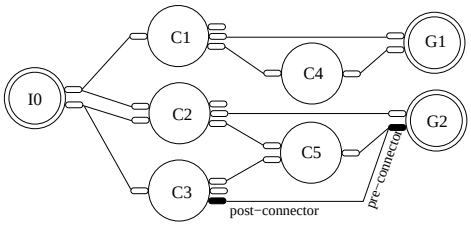

Fig. 3. Schematic Learning Graph.

Figure 3 shows a simple learning graph example that uses five (atomic) cases retrieved from the Case Base.

In this example, two concepts (G_1 and G_2) will be learned. From this learning graph, the next solutions (or *learning paths*) are possible:

- concepts to be learned: $G_1 \wedge G_2$
- **solution 1**: I_0—C_1—C_4—$G_1 \wedge I_0$—C_2—C_5—G_2
- **solution 2**: I_0—C_1—C_4—$G_1 \wedge I_0$—C_2—C_3—G_2
- **solution 3**: I_0—C_1—C_4—$G_1 \wedge I_0$—C_2—C_3—C_5—G_2

The following learning paths: I_0—C_1—G_1, I_0—C_2—G_2, or I_0—C_3—G_2 cannot be considered as correct solutions because if they were considered, some pre-connectors in the goal nodes (G_1 and G_2) will be unconnected. This means that some concepts that the user needs to achieve for his/her (learning) goals, will never be acquired. The post-connectors that are not connected in the learning graph represent those extra concepts that have been acquired by the student in his/her learning process.

3.2 Case Model

Cases can be represented in a variety of forms using the full range of AI representational formalisms including frames, objects, predicates, semantic nets and rules.

The frame/object representation is currently used by the majority of CBR software. A case is a contextualised piece of knowledge representing an experience. It contains the past lesson that is the content of the case and those features, or characteristics, in which the lesson can be used [8]. Typically a case comprises:

- The *problem* that describes the state of the world when the case occurred. In our approach several keywords are used as pre-connectors to represent what concepts are necessary to understand the information stored in the case.
- The *solution* which states the derived solution to that problem. In our approach the case stores the information related to a particular programming topic.
- And/or the *outcome* which describe the state of the world after the case is applied. In our approach several keywords that represent those concept that have been learned by the user will be used as post-connectors.

The architecture of the Case-Based subsystem is shown in Figure 4. This subsystem is implemented using the following modules:

- *Case Creator Tool.* This tool allows the engineer building the initial *atomic* cases that represent all the available knowledge about a particular topic. It also allows including the content of the case, the keywords used to characterize the store information in the case, the learning connectors that are "learned" by the user once the content is studied, and the complexity of the case.
- *NLP module.* Although initially some cases characteristics like the keywords, or *connector* will be included by the engineer, a NLP analysis will be achieved by this module to suggest the characteristics that could represent the case.
- *Retrieving module.* This module implements one or several matching functions that are used to retrieve the most promising stored cases.

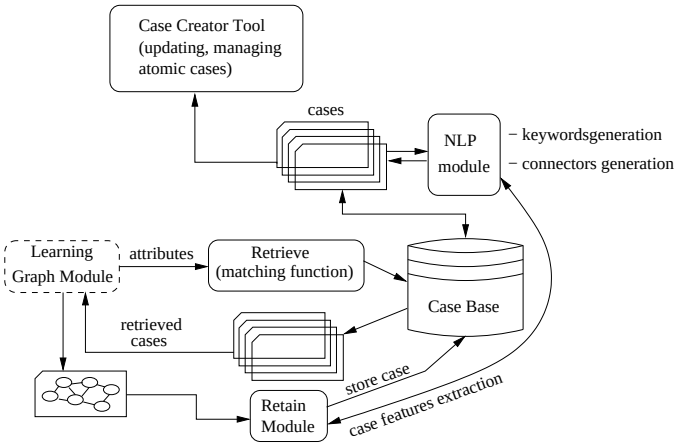

Fig. 4. CBR subsystem in DynJAQ.

– *Retain module*. Once one or several solutions are successfully found, they will be stored as new cases. Therefore, the NLP module will be used to obtain the keywords and connectors that will be used to represent the case.

The Case Base is made by two different types of cases: atomic and complex.

– Atomic cases: are built by the engineer, and represent the specific knowledge about a particular topic. For instance, if we consider the problem of learning Java programming, these atomic cases represent the specific information about a particular topic in the language, i.e. an atomic case could be created to provide information about *how to define a variable*, other could represent *how to define a method* in Java Language, etc. Any atomic case is built by the next components:
  - *Content*: stores the knowledge (in natural language) about one or several concepts.
  - *Keywords*: represent a list of words that represent the semantic information stored in the content of the case.
  - *Complexity*: this attribute is actually fixed by the engineer and represents the complexity of the concept, or concepts stored in the case.
  - *Connectors*: pre and post connectors represent the learned concepts by the students if the content of the case is completely understandable.
  - *Granularity*: represents how detailed is the information stored by the case. The granularity is fixed by the engineer when builds the case. It is possible to measure the granularity of a particular atomic case using the number of keywords and connectors used by the case. Therefore, a thin granularity will use few keywords and connectors, because the knowledge stored is very specific. However, if a rough granularity is used the number of keywords and connectors will grow up.

The concept of granularity is very important because this feature could not be homogeneous. Therefore it is possible to store different atomic cases with different granularities. Our approach allows managing different granularities into a common solution. The number of stored cases in the Knowledge Base are related to this feature.

- Complex cases: are built by means of the user/system interaction. When any student provides a question to the system, one or several *learning graphs* will be implemented. The final user evaluation is used to decide if the new case will be stored (or rejected) in the Case Base. Initially all the cases stored in the Case Base are atomic. However, the interaction with the users modifies both the number and complexity of the stored cases. The components of this new type of cases are:
  - *Content*: is built by the content of all the atomic cases.
  - *Keywords of the complex case*: are the union of all the (different) keywords of the atomic cases.
  - *Complexity of a complex case*: is calculated as the maximum atomic case complexity stored in the learning graph.
  - *Connectors*: the pre/post-connectors are the union of all the atomic cases in the learning graph.
  - *Granularity*: is automatically obtained by adding the different granularities of those atomic cases that build the case.

4 DynJAQ: A Dynamic Web FAQ System

This section describes how our approach has been instantiated into a particular implementation. We have designed and implemented a Dynamic Web FAQ System named DynJAQ (Dynamic Java Asked Question). DynJAQ is able to solve questions about how to programm in Java and it can be used like a Java-related FAQ repository. However, the answer(s) given by DynJAQ will be adapted to the user characteristics (like his/her programming level). Figure 5 shows the different modules that implements the DynJAQ architecture. The functionality of these modules can be summarized as follows:

- *User/system interaction.* The interaction between the users and the system has been carried out using Web Services technologies. The system uses a module (called $LearningGraph_{TO}HTML$) that is the responsible to generate an user-friendly representation (an HTML guide) for each possible solution. The set of Graphical User Interfaces (GUIs) provides the following information:
  - The question that represents the concepts that the user wish to learn about Java Programming.
  - The expertise knowledge, or skill programming of the user.
  - The maximum number of possible solutions (answers) found for his/her question.

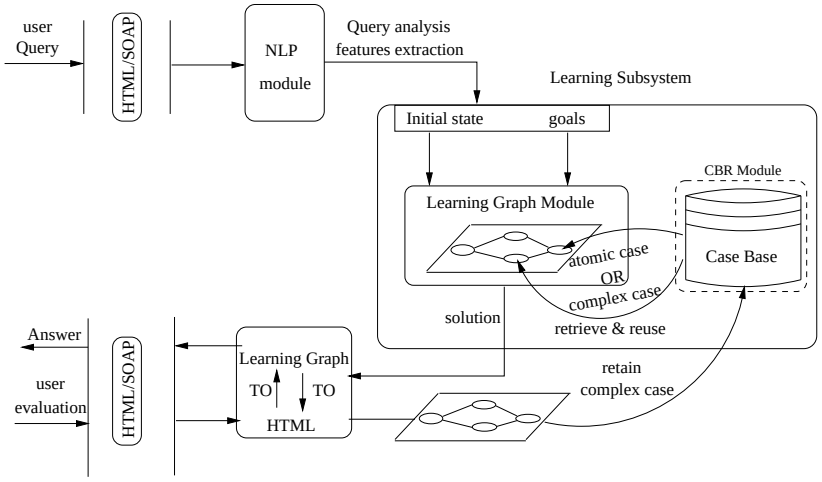

Fig. 5. DynJAQ architecture.

- *NLP module*: performs the analysis of the query. Our approach uses a simple NLP technique to extract the keywords from the user query (only a list of stop words are used to extract the keywords from the question). This module is used to extract other features (like the user characteristics) from the question. This module provides the necessary information to build the initial and final nodes.
- *Learning Subsystem*: has been implemented using two related submodules:
  - *Learning graph module*: a hierarchical graph is built using the information obtained from the NLP module and the CBR module.
  - *CBR module*: is used by the previous module to retrieve the most promising cases stored in the Case Base.

Finally, the interaction with the user is used to learn those solutions that he/she marks as a success. The graphs used to build those successful solutions will be stored as new cases in the Case Base.

Figure 6 (a) and (b) show a possible input to the system given by a beginner (Java programmer) user and the request given by the system. When a question is processed by the NLP module in DynJAQ, the input information is translated into:

- The expertise level of the user: used as the initial node in the graph.
- The extracted keywords from the question: used as target goals (or final nodes) in the graph.

Using both type of nodes, a new learning graph will be generated by the Learning Module. This learning graph will be used to represent the different solutions (or paths from the initial level of knowledge to those goal concepts that the user wishes to learn) that the system is able to find.

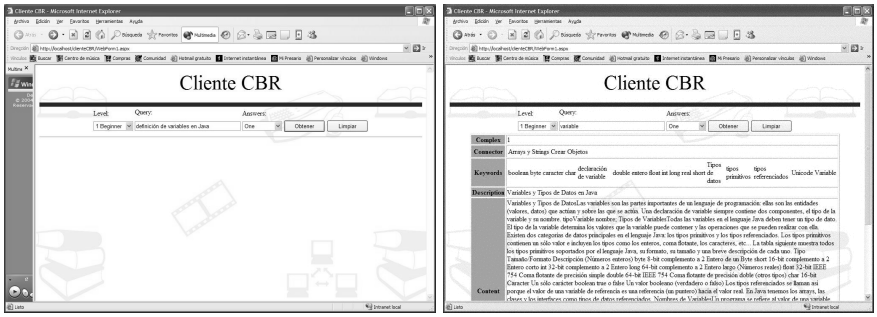

Fig. 6. (a) Question about how to define a variable in Java Language programming,(b) DynJAQ answer for a simple question.

For instance, let us suppose that the question, shown in Figure 7 is given to DynJAQ. In this example, the general goal can ge represented as: **learn-to-define (classes,variables,constants)**. This general *learning-goal* can be decomposed into three more specific goals (these goals could be dependent). To complete each of those tasks, the Case Base is accessed to retrieve knowledge (cases) that could bind the preconditions and postconditions of the tasks.

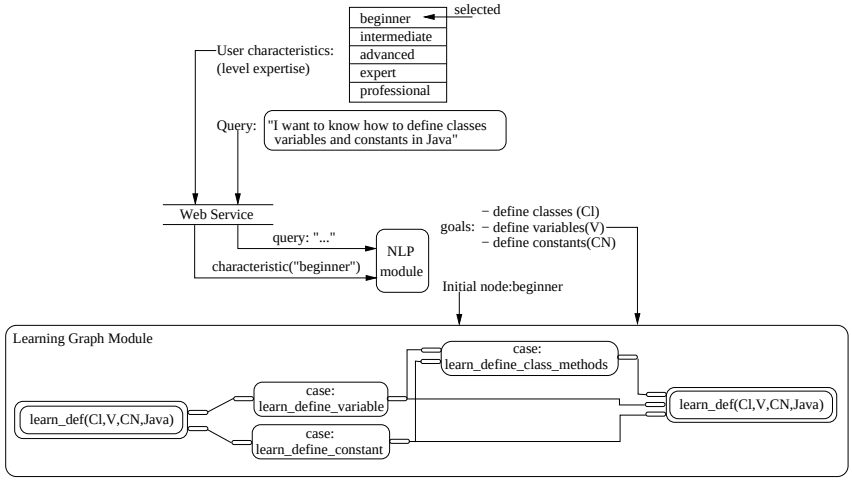

Fig. 7. Question given to DynJAQ.

The preconditions and postconditions of each operator will be related to a set of keywords that represent both, the *knowledge* that is necessary for the user to learn the concept represented by the node (preconditions), and the *learned concepts* if the node is applied (postconditions).

5 Conclusions

The main contribution of this work is to define a hierarchical knowledge representation that can be applied to build a new type of FAQ systems based on techniques like NLP [6], CBR [1, 2], or Web Services in order to allow the implementation of flexible question answering systems.

With the hierarchical knowledge representation proposed in this paper in a particular kind of Web systems such as the FAQ Systems, features like adaptability or flexibility, can be improved if several AI techniques are adequately used. We have used in DynJAQ those techniques as follows:

- NLP techniques are used to analyse the user query (to *extract* the keywords from this query), and to manage the *keywords* and *connectors* that are stored in the Case Base.
- CBR is used to represent and manage the knowledge about a particular topic or domain.
- The hierarchical learning graphs are used to build an adaptive solution to the user question. Using the keywords (and other user information) like "learning-goals", a new graph can be generated for each query.
- Web Services technologies are used (HTML, XML, SOAP, etc...) to implement an interoperable and flexible Web System.

References

1. A. Aamodt and E. Plaza. Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AICom-Artificial Intelligence Communications. IOS Press*, 7(1):39–59, 1994.
2. D. W. Aha, L. Breslow, and H. Muñoz-Avila. Conversational case-based reasoning. *To appear in Applied Intelligence*, 2000.
3. R. Burke, K. Hammond, V. Kulyukin, S. Lytinen, N. Tomuro, and S. Schoenberg. Natural language processing in the faq finder system: Results and prospects, 1997.
4. R. D. Burke, K. J. Hammond, and E. Cooper. Knowledge-based information retrieval from semi-structured text. In *In AAAI Workshop on Internet-based Information Systems*, pages 9–15. AAAI, 1996.
5. R. D. Burke, K. J. Hammond, V. A. Kulyukin, S. L. Lytinen, N. Tomuro, and S. Schoenberg. Question answering from frequently asked question files: Experiences with the FAQ finder system. *AI Magazine*, 18(2):57–66, 1997.
6. C. L. A. Clarke, G. V. Cormack, and T. R. Lynam. Exploiting redundancy in question answering. In *Research and Development in Information Retrieval*, pages 358–365, 2001.
7. R. Davis, H. Shrobe, and P. Szolovits. What is a knowledge representation? - an introductory critical paper. *AI Magazine*, 14(1):17–33, 1993.
8. J. Kolodner. *Case-Based Reasoning.* Morgan Kaufmann, 1993.
9. C. C. T. Kwok, O. Etzioni, and D. S. Weld. Scaling question answering to the web. In *World Wide Web*, pages 150–161, 2001.