DynJAQ: An Adaptive and Flexible Dynamic FAQ System

David Camacho,[1,†] Maria Dolores R.-Moreno[2,*]
[1]*Computer Science Department, Universidad Autónoma de Madrid, C/Francisco Tomás y Valiente, n° 11, 28049, Ciudad Universitaria de Cantoblanco, Madrid, Spain*
[2]*Departamento de Automática, Universidad de Alcalá, Ctra. Madrid-Barcelona, Km. 33,6, 28805, Alcalá de Henares, Madrid, Spain*

This article presents a new type of Frequently Asked Questions (FAQ) System, called DynJAQ (Dynamic Java Asked Questions) that has been designed with the purpose of making learning more appealing to beginner students of engineering disciplines and overcome the inconvenience of these systems. DynJAQ is able to generate dynamically several HTML guides that can be used to answer any possible question about a particular programming language (Java), although it can be easily extended to any other topic. DynJAQ integrates 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 it 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 management of these cases allows enriching the knowledge base.

1. INTRODUCTION

One of the difficulties for beginner students[1] in engineering disciplines is finding and processing the large amount of information that they need in order to prepare their related matters. Depending on their expertise, the same questions come up frequently in their learning phase. When any student needs to learn a particular concept, that is, 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 common, it is possible to build a Frequently Asked Questions (FAQ) repository to answer these questions.

*Author to whom all correspondence should be addressed: e-mail: malola@email.arc.nasa.gov.
†e-mail: david.camacho@uam.es.

INTERNATIONAL JOURNAL OF INTELLIGENT SYSTEMS, VOL. 22, 303–318 (2007)
 Published online in Wiley InterScience (www.interscience.wiley.com). • DOI 10.1002/int.20198

A common solution is to use the Web as a FAQ repository to store the most common questions (or problems) detected by the students and the solution (or solutions) proposed to solve these problems to avoid continuously repeating the same explanations. However, all of these FAQ systems are static, and they cannot add new topics (question and its related answer) into their repositories. DynJAQ is a new approach that uses artificial intelligence (AI) techniques that help to implement *adaptive* and *dynamic* FAQs. Using several AI techniques, such as Natural Language Processing, the user is able to ask questions such as: "I want to know how to create classes and methods, define objects and method calls in Java," and DynJAQ gives to the user one or several solutions for this particular question *adapted* to his/her skills. The system defines a method of knowledge representation[2,3] able to integrate predefined knowledge about a particular topic (cases) into a common solution. These solutions will be built using this knowledge as the basic (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 users' skills. DynJAQ is part of a collaborative project between two universities that aims at producing new innovative experiences in learning.

The article is structured as follows: Section 2 describes the knowledge structure that allows implementing an *adaptive* and *dynamic* FAQ system. Section 3 shows in detail the features and implementation of DynJAQ. Then, Section 4 shows the related work. Finally, Section 5 summarizes the conclusions and future work.

2. THE DynJAQ KNOWLEDGE REPRESENTATION

This section describes how to combine a graph-based representation and case-based knowledge into a hierarchical representation that can be used to build flexible *Question Answering* (QA) systems.[4] A QA system directly provides an answer to the user question by accessing and consulting its knowledge base; these kinds of systems are related to the research area of Natural Language Processing (NLP).[5] In our approach we have used Case-Based Reasoning to implement a dynamic and interactive FAQ system.

2.1. 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 case-based reasoning (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 preconnectors and postconnectors to represent which concepts are necessary to understand the knowledge stored in the node (preconnectors) and what concepts are acquired if this node is learned by the user (postconnectors). Those connectors are used to define the *transitions* between the nodes in the graph.

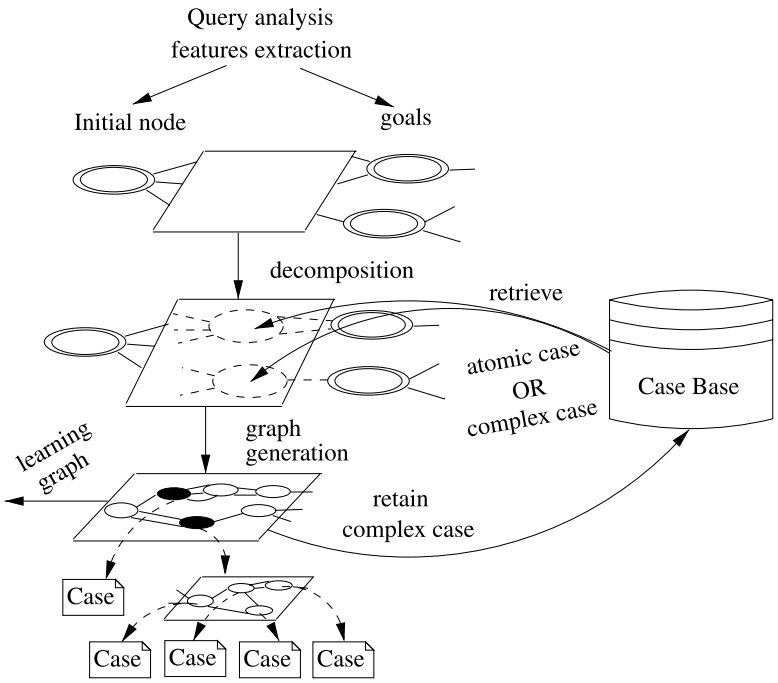


Figure 1. Learning graph representation.

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 1 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. This algorithm is summarized in Figure 2.

Let us suppose the following example: A beginner student wants to learn to program threads in Java (a question like "I wish to know how to declare and use threads in Java" will be given to the DynJAQ system). From this question two learning goals are acquired (G_1 = learn to declare threads and G_2 = learn to use (methods) threads). DynJAQ is able to provide a complete tutorial (or tutorials) that can be used by the user such as a customized guide to solve his/her problem. Therefore, in our example, the student will learn to program the next Java sentences:

```
//The following sentence declares a thread in Java Thread
myObject = new Thread();

//This sentence calls a predefined method of the object
myObject.run();
```

(1) Using the user query (and other user features), the initial and final nodes are defined (see Section 3).
(2) While a path does not exist from the final nodes to the initial node, do:
  (2.1) Retrieve from the Base Case all the cases that match with all the pre-connectors of the final nodes.
  (2.2) From the retrieved cases select those whose pre-connectors connect completely all the post-connectors in the previous node.
    (2.2.1) If there is 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 a case does not exist that matches with all the pre-connectors for a particular node, retrieve a set of nodes that partially matches with those connectors, until all the pre-connectors are connected.
    (2.3.1) For each selected node verify that all of its pre-connectors are connected; if not connected, 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).

Figure 2. Learning graph algorithm generation.

Following the example, Table I shows some possible cases (and their related preconnectors/postconnectors) that could be retrieved from the Case Base. These cases represent some stored concepts about Java Programming. For instance, Case 4 stores a description about how it is possible to declare a Thread in Java (post-connector = declare thread); therefore the student must understand the following Java sentence: `Thread myObject = new Thread();`. However, to understand correctly how to declare previously this type of object it is necessary to know how to *build* any object in Java (preconnector = *new* operator).

Using previous retrieved cases (from C_1 to C_5), the learning goals (G_1 and G_2), and the initial student features (I_0), DynJAQ builds the learning graph shown in Figure 3.

From the previous learning graph, the next solutions (or *learning paths*) are possible:

- **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 preconnectors 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. For instance, in the first rejected learning path (I_0—C_1—G_1), it is not possible to declare a thread if previously the student does not know how to

Table I. Possibles cases retrieved from a specialized Java Programming Case Base, to learn how to declare and use threads in Java.

Case	Preconnector	Postconnector
C_1	Declare object	Define object Declare variable *new* operator
C_2	Declare method Declare object	List parameters Call object Call method
C_3	Declare method	Declare paremeter Define method List parameters
C_4	*new* operator	Declare thread
C_5	*return types* List parameters	Define method
G_1	Define object Declare thread	—
G_2	Define method Call method	—
I_0	—	Define object Declear method

build an object (using the *new* operator). Therefore, to declare threads correctly, case C_4 must be studied to achieve the G_1 goal.

The postconnectors that are not connected in the learning graph (i.e., the postconnector "Declare variable" in case C_1) represent those extra concepts that have been acquired by the student in his/her learning process.

2.2. 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,

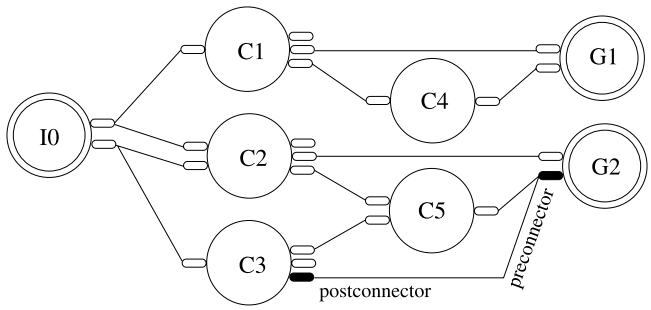


Figure 3. Schematic learning graph.

and rules. The frame/object representation is currently used by the majority of CBR software. A case is a contextualized piece of knowledge representing an experience. It contains the past lesson that is the content of the case and the context in which the lesson can be used.[6] Typically a case is comprised as follows:

- The *problem* that describes the state of the world when the case occurred. In our approach, several keywords are used as preconnectors 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 describes 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 postconnectors.

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. Figure 5 shows an atomic case that stores information about the topics "variables and basic types in Java."
- *NLP module*. Although initially some case characteristics like the keywords or *connector* will be included by the engineer, an NLP analysis will be achieved by this module to suggest the characteristics that could represent the case.
- *Retrieving module*. This module implements a matching (or similarity) function that is used to retrieve the most promising stored cases. The similarity function (S_f) that is

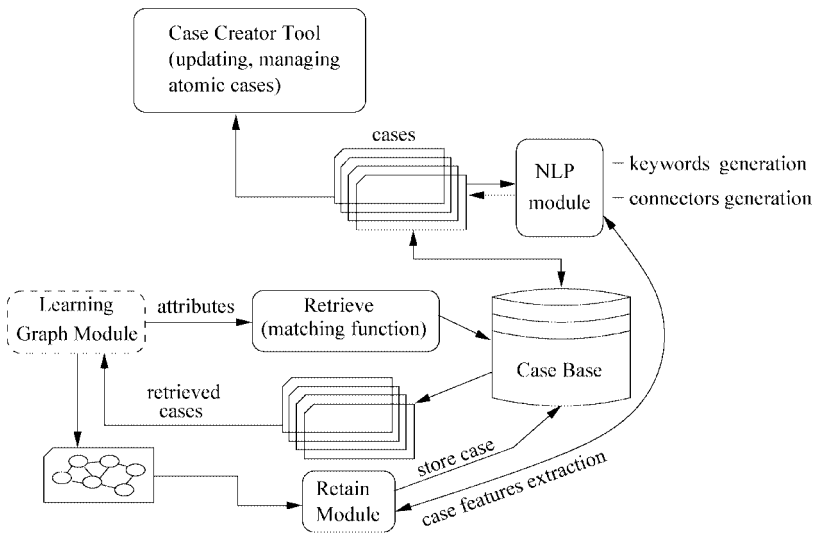


Figure 4. CBR subsystem in DynJAQ.

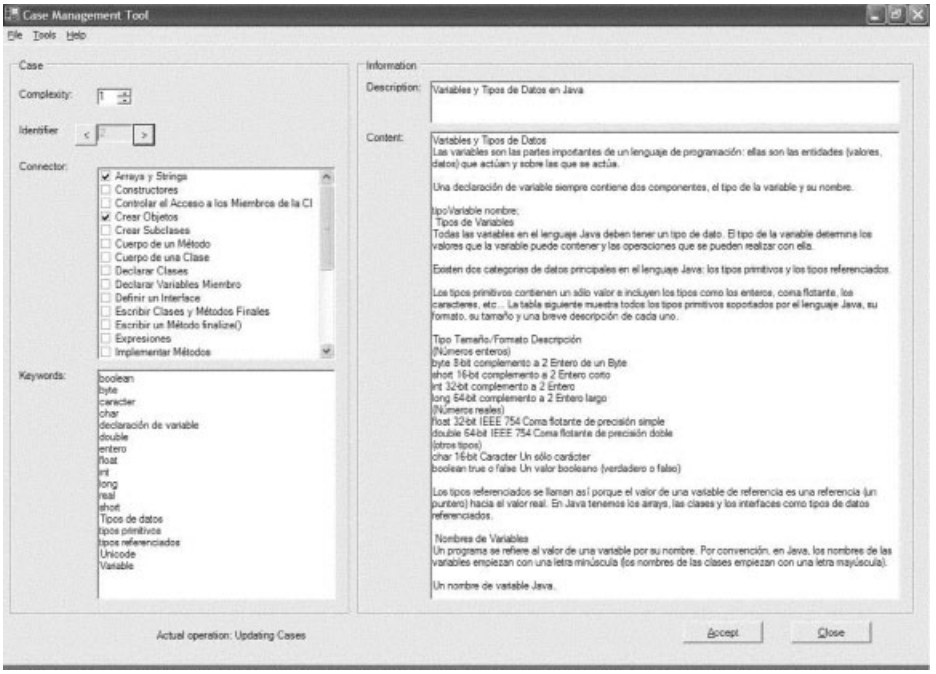


Figure 5. Atomic case definition.

used to retrieve the cases stored from the Case Base allows the selection of the most promising set of cases that later will be used as nodes in the learning graph. Equation (1) shows the S_f used in DynJAQ. For each case a similarity value is calculated; those cases with maximum values are selected to build the set of retrieved cases. The similarity value is obtained using the relationship between the keywords given by the user and the keywords that are used to represent the contents of the case. The Hits value is an ordinal value that represents the intersection between the number of user keywords and case keywords; in the best case (the case contains all the user keywords) the Hits number will be equal to the Number (user-keywords), and therefore the similarity ($S_f(C_i)$) value for this case (C_i) will be 1. In the worst case (no user keyword is stored in the selected case) the number of Hits, and the $S_f(C_i)$ value, will be 0. Therefore, the $S_f(C_i)$ value measures how well the selected case fits the user necessities.

$$S_f(C_i) = Hits(user - keywords \cap case - keywords)/Number(user - keywords) \quad (1)$$

- *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; that is, an atomic case could be created to provide information about *how to define a variable*, another could represent *how to define a method* in Java language, and so forth. 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 postconnectors represent the concepts learned by the students if the content of the case is completely understandable.
- *Granularity* represents how detailed the information is that is stored by the case. The granularity is fixed by the engineer when he or she 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 increase.

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 the following:
  - *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/postconnectors 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.

3. 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 program in Java, and it can be used like a Java-related FAQ repository. However, the answer(s) given by DynJAQ will be adapted to user characteristics (like his/her programming level). Figure 6 shows the different modules that implement 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 responsible for generating a user-friendly representation (an

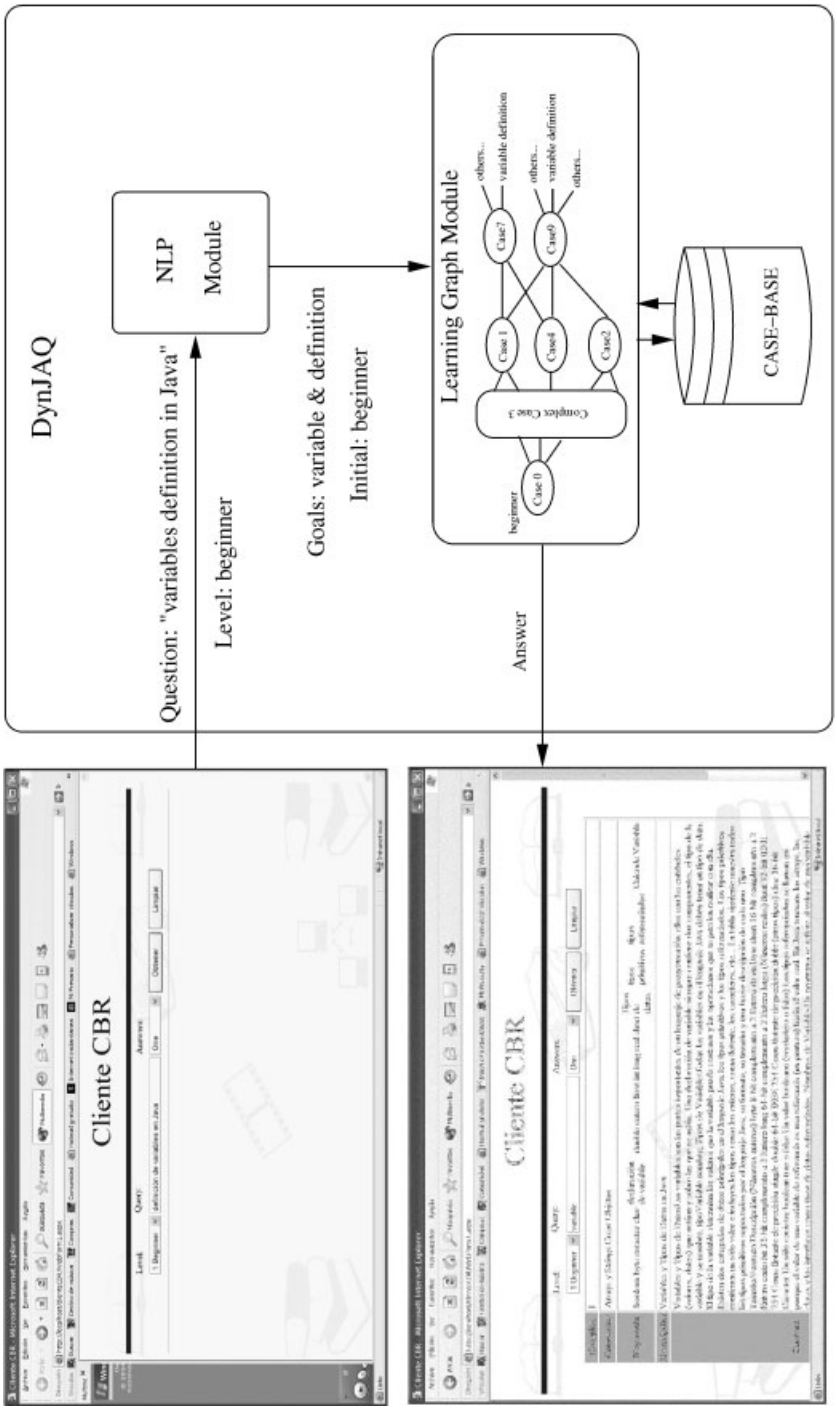


Figure 6. DynJAQ architecture.

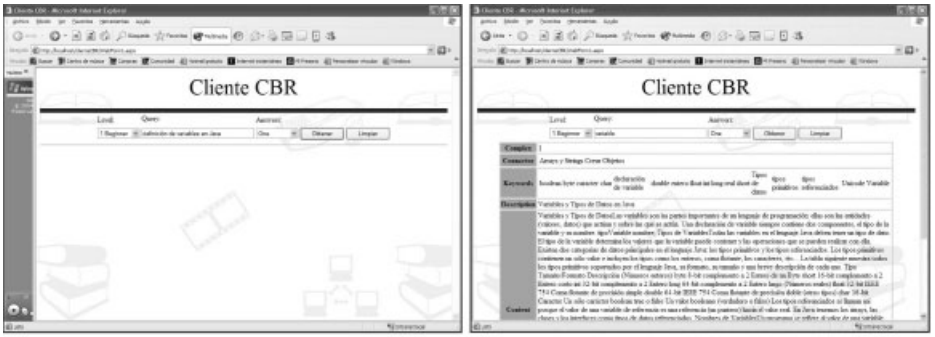
HTML guide) for each possible solution. The set of Graphical User Interfaces (GUIs) provides the following information:
  – the question that represents the concepts that the user wishes to learn about Java Programming
  – the expertise knowledge or skill programming of the user
  – the maximum number of possible solutions (answers) found for his/her question.
- *NLP module*. This 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 user characteristics) from the question. This module provides the necessary information to build the initial and final nodes.
- *Learning subsystem*. The 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*: This 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 7a, b shows 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.



(a) (b)

Figure 7. (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 8, is given to DynJAQ. In this example, the general goal can be 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 preconnectors and postconnectors of the cases.

The preconnectors and postconnectors for each case are 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 (preconnectors), and the *learned concepts* if the node is applied (postconnectors).

Figure 9 shows a simple example of some possible preconditions and postconditions for a particular node.

Following this example, each of the decomposed goals in the hierarchical network needs to be completed with specific information. As Figure 10 shows, the example shown in previous figure can be translated into a **case-based graph**.

4. RELATED WORK

This section addresses some systems related to our approach. We briefly describe the main features of those systems in areas such as FAQ Web systems, Question Answering systems, or CBR systems and compare them against DynJAQ.

4.1. FAQ Web Systems

Usually a FAQ works in the following way: Any user can consult a preexisting list of questions with their answers in order to find a similar question that can answer his/her own problem. The user has the responsibility of analyzing all the items in the FAQ (usually searching through previous questions asked by other users) and finding the most similar solutions and "reusing" or "adapting" them for his/her problem. There are a huge number of Web sites that provide FAQ repositories. Some interesting examples are the following:

- The Dynamic FAQ Database (http://products.dynamicwebdevelopers.com) is a Web application that allows users to construct a Frequently Asked Questions Database for a particular Web site. 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.
- A Web site 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 users 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 as follows:

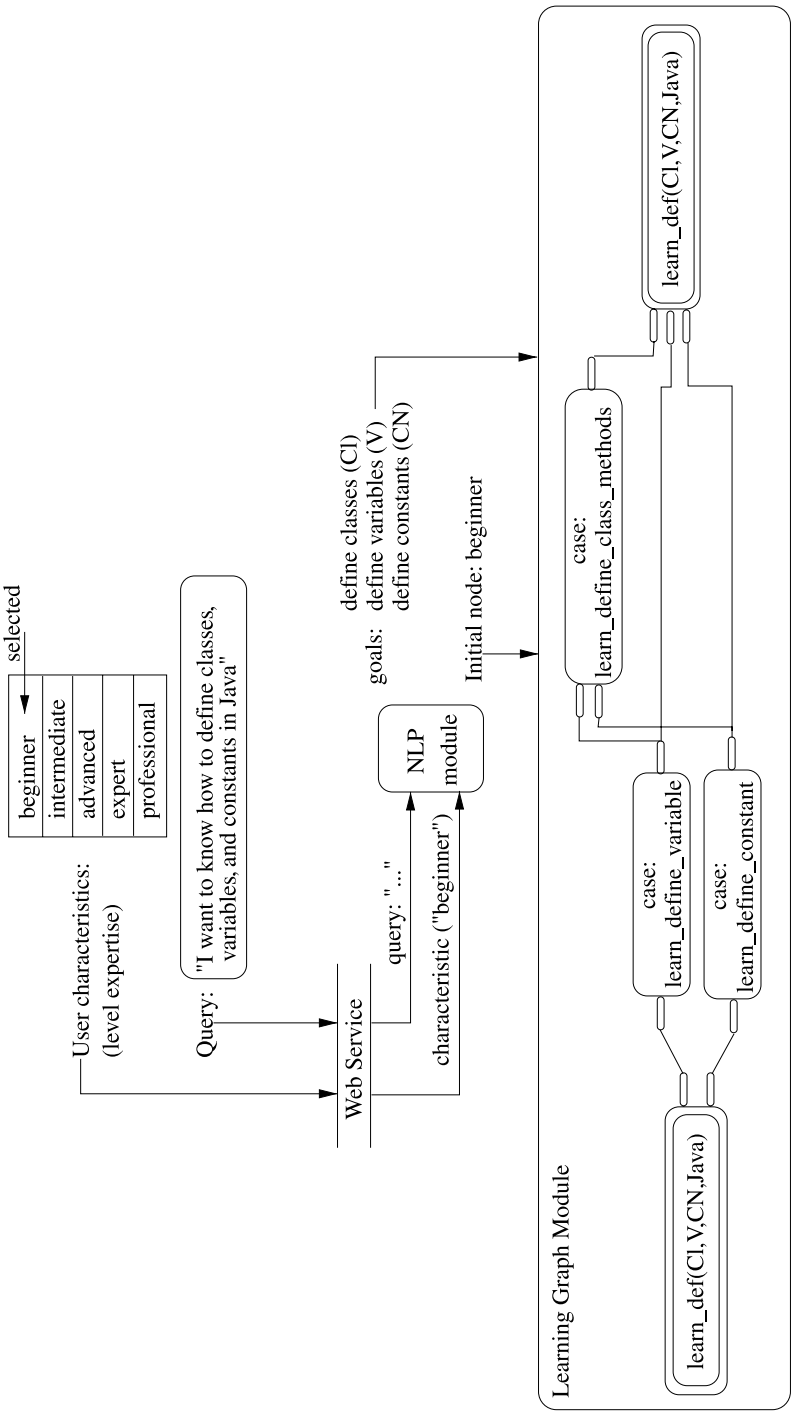


Figure 8. Question given to DynJAQ.

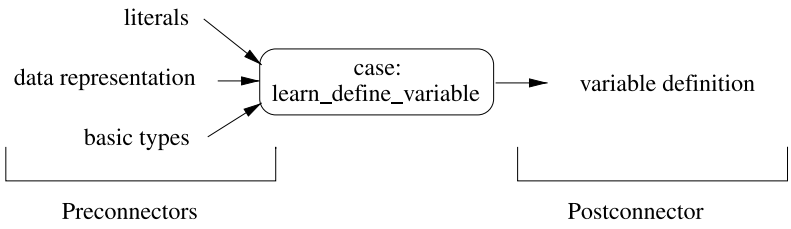


Figure 9. Preconnectors and postconnectors in a node.

(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 overspecific and the user could not correctly understand the solution proposed or only in a very general way so the user does not find what she or he is really looking for.
(2) Usually the user must select among some existing questions to find the most appropriate.
(3) If a Web system is implemented using only a static repository of solutions, this repository will increase in size quickly and the number of documents retrieved could be very large. Therefore, the users might not find the information or it could be very hard.
(4) No FAQ system takes into account the user features, or skills, about a particular topic.
(5) These kind of systems are not flexible because they store a set of predefined problems and their related solutions.

DynJAQ has been designed to overcome the problems described about simple FAQ systems: It takes into account the user skills, adapting the solution to his/her knowledge; it is flexible as it generates dynamically the solutions. Finally, it does not use a static repository, so finding the solution is relaxed in our system.

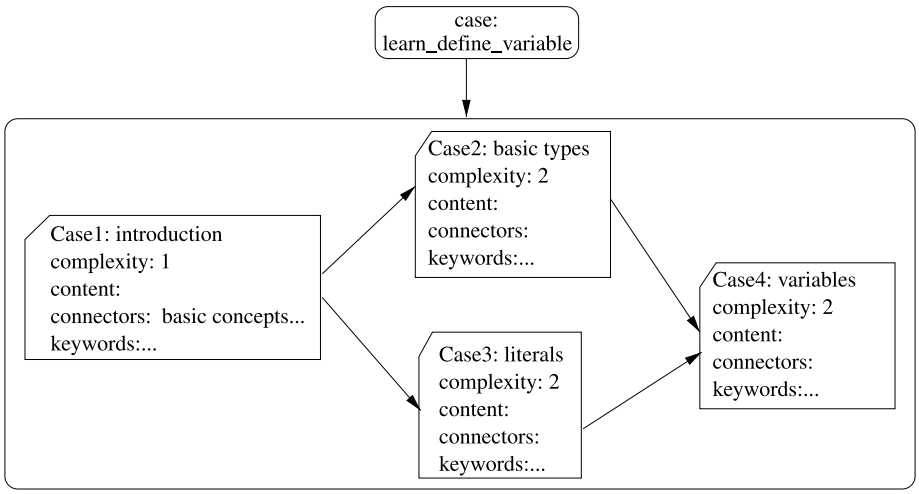


Figure 10. Decomposition of a complex node into simple (atomic) cases.

4.2. Question Answering Systems

There is an important research work related to the *question answering* (QA) systems.[4] A QA system directly provides 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).[5]

In recent years, and due to the evolution of the Web, a new interest in the application of QA systems to the Web has arisen. In Ref. 7 what is required to implement a Web-based QA system is defined. 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 to retrieve relevant documents from the collection, that is, 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.[4,8,9] FAQ Finder (http://josquin.cti.depaul.edu/~rburke/research/faqfinder.htm) 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 precompile much of the system knowledge; thereby 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 made easier by using a general graph-based search technique integrated with domain dependent case-based knowledge.

4.3. Case-Based Reasoning Systems

Case-based reasoning (CBR)[6,10,11] 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 11 shows the processes involved in CBR. This general CBR cycle may be described by the following four processes[10]:

- 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.

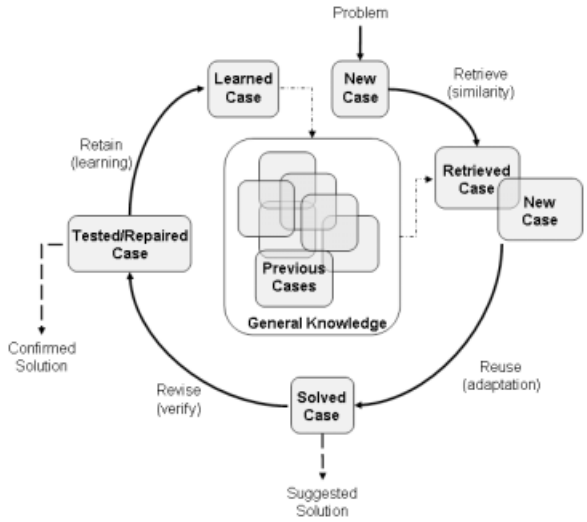


Figure 11. The CBR cycle.

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 kinds of cases, atomic and complex, as explained in Section 2.

5. CONCLUSIONS AND FUTURE WORK

In this article, we have presented an adaptable and dynamic FAQ system, called DynJAQ. FAQ systems have the inconvenience of being very rigid, and they do not consider the users' skills. In our system, we have added features like adaptability or flexibility (that are not present in FAQ systems), integrating adequately several AI techniques. We have used in DynJAQ those techniques as follows:

- NLP techniques are used to analyze 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 (XML, SOAP, etc.) to implement an interoperable and flexible Web system.

The main contribution of this work is to define a general hierarchical knowledge representation that can be applied to build a new type of FAQ system based on techniques like NLP,[5] CBR,[10,11] or Web services to allow the implementation of flexible FAQ and Question Answering systems.

Currently we are working in two different ways to improve the DynJAQ system:

(1) We are extending the NLP module both to gain flexibility in the user/system communication and to automatically extract the keywords and connectors from the contents in an atomic case.
(2) We are integrating this flexible and customizable FAQ system into a Learning Management System, called *e-Go*, that currently has being deployed in several Spanish universities.

Acknowledgments

This work has been partially funded by the innovative learning experience projects of Universidad de Alcalá (UAH PI2005/084). We want to thank Alberto López and César Castro for their help in the implementation of DynJAQ.

References

1. Reid AT. Perspectives on computers in education: The promise, the pain, the prospect. Active Learning. Oxford, UK: CTI Support Service; December 1994.
2. Davis R, Shrobe H, Szolovits P. What is a knowledge representation? An introductory critical paper. AI Mag 1993;14:17–33.
3. Sowa JF. Knowledge representation: Logical, philosophical, and computational foundations. Pacific Grove, CA: Brooks Cole Publishing Co.; 1999.
4. Burke RD, Hammond KJ, Kulyukin VA, Lytinen SL, Tomuro N, Schoenberg S. Question answering from frequently asked question files: Experiences with the FAQ finder system. AI Mag 1997;18:57–66.
5. Clarke CLA, Cormack GV, Lynam TR. Exploiting redundancy in question answering. In: Research and development in information retrieval. SIGIR: ACM Special Interest Group on Information Retrieval, SIGIR'2001. New York: ACM Press; 2001. pp 358–365.
6. Kolodner J. Case-based reasoning. San Francisco, CA: Morgan Kaufmann; 1993.
7. Kwok CCT, Etzioni O, Weld DS. Scaling question answering to the web. In: Proc Tenth World Wide Web Conf. New York: ACM Press; 2001. pp 150–161.
8. Burke RD, Hammond KJ, Kulyukin VA, Lytinen SL, Tomuro N, Schoenberg S. Natural language processing in the faq finder system: Results and prospects. Proc AAAI Spring Symp on Natural Language Processing for the World Wide Web. Menlo Park, CA: AAAI Press; 1997.
9. Burke RD, Hammond KJ, Cooper E. Knowledge-based information retrieval from semi-structured text. In: AAAI Workshop on Internet-based Information Systems. Menlo Park, CA: AAAI; 1996. pp 9–15.
10. Aamodt A, Plaza E. Case-based reasoning: Foundational issues, methodological variations, and system approaches. AI Comm Eur J Artif Intell 1994;7:39–59.
11. Aha DW, Breslow L, Muoz-Avila H. Conversational case-based reasoning. Appl Intell 2001;14:9–32.