A rule-based semi-automated approach to building Natural Language Question Answering (NLQA) systems

Kaushik Krishnasamy
Intelligent Systems Application Center
College of Engineering
Temple University
Philadelphia, PA 19122 USA
kaushik@temple.edu

Brian P. Butz
Intelligent Systems Application Center
College of Engineering
Temple University
Philadelphia, PA 19122 USA
bpbutz@temple.edu

Michael Duarte
Intelligent Systems Application Center
College of Engineering
Temple University
Philadelphia, PA 19122 USA
mduarte@temple.edu

Abstract

This paper presents a rule-based approach to natural language question answering that can be easily implemented for any domain. We discuss the framework in the context of a National Science Foundation funded project - Universal Virtual Laboratory (UVL). UVL is a virtual electrical engineering (EE) laboratory for able and disabled individuals to construct, simulate and understand the characteristics of basic electrical circuits. Like any real-life laboratory, the UVL has a teaching assistant (TA), to whom students can ask questions related to basic EE and on using the laboratory in general. The virtual TA is built using the framework that is discussed in the paper. The framework relies on careful design rather than complex grammar or statistics. It can be used for building rapid language comprehension applications specific to a particular domain by using the appropriate heuristics/keywords of that domain. Also introduced is the automation tool that can be used to implement the framework for any domain.

Introduction

According to the Center for Disease Control (CDC 1995) there are 13.6 million individuals who have limited hand use and another 16.3 million who have mobility limitations. In the field of science and engineering, there are approximately 109,700 persons with motor disabilities employed in the United States (NSF 1997). Also, approximately 31,300 students with motor disabilities were registered in science and engineering programs in 1995 (NCES 1996).

The Universal Virtual Laboratory (UVL) (Duarte and Butz 2002) provides a disabled student with motor disabilities access to realistic laboratory experience. The main goal of the UVL is to create an environment similar to a real electrical engineering laboratory, and to offer the user (a disabled or physically able individual) a way to learn the different aspects of instrumentation and circuitry. The UVL achieves this by using several existing Windows® applications. The instruments available to the user are a power-supply, a function generator, an oscilloscope, a spectrum analyzer and a digital multimeter. The instruments are connected to the holes on the breadboard when the user clicks[1] on the instrument and the coordinates of the hole where he/she wishes to place the component.

The components in the UVL include resistors, capacitors, inductors, diodes, transistors, switches and jumper wires. The coordinates are the letters and numbers seen on the breadboard (Figure 1).

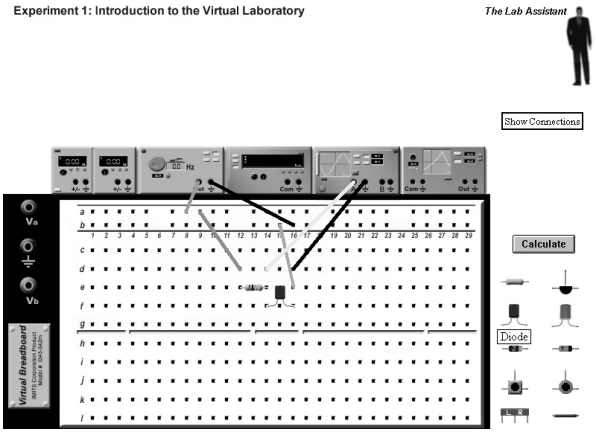

Figure 1: The Universal Virtual Laboratory

After a circuit has been built, the user hits the "Calculate" button to simulate the circuit behavior. The calculate button activates the circuit analysis program 'Pspice' which validates the user built circuit. If the circuit is connected properly (according to Pspice requirements), the output devices (multimeter / oscilloscope / spectrum analyzer) display the result. The current research is an effort to complement the above endeavor by equipping the laboratory with an intelligent TA (NLQA framework), who can respond to plain English queries posed by students when they use the lab.

The Natural Language Framework

The NLQA framework or the shell discussed here is domain independent. The NLQA shell is "fitted" with the

[1] "Clicking" the mouse may also be performed verbally by students with a mobility disability

vocabulary of the domain, its rule base files, its hierarchy, etc. created by the domain expert for a specific subject area. The framework is unaltered for any domain in which it has to work. The framework's keyword-based approach along with the shell-'information base' structure builds on the principles used in ELIZA (Weizenbaum 1966) and CHAT (Whalen and Patrick 1989). The following sections describe the various domain specific parameters, special features of this framework and their interactions.

Domain Vocabulary

The framework requires the vocabulary of the domain under consideration as one of its inputs. For example, "resistor", "measure", "voltage", "dc", etc. are keywords (or words of consequence) in a domain for electrical circuits. The user query is expected to contain at least one keyword. Keywords are selected by a domain expert, usually the subject content provider, who oversees the inclusion of material within the software system. The tool that the domain expert can use to build the vocabulary is discussed later.

Keywords are stored in level-node configurations, where levels are conceptually distinct entities, while nodes are related entities, yet disparate within the larger "level" context. Each of these nodes has a list of synonyms. These synonyms are referred to by the combination of the level number and the node numbers under which they fall. This maps a large set of similar/synonym keywords to a single level-node address. The level-node-synonym hierarchy lends itself easily to be implemented using XML.

In this NLQA system, levels and nodes are used to store the domain vocabulary. Level 1 stores certain nouns which, in this domain, consist of circuit elements (e.g. resistors, capacitors, etc.) or instruments (e.g. multimeter, function generator, etc.). Each element or instrument occupies a node within level 1. Consequently "resistor" (and its synonyms) might occupy level1-node1 represented by "l1n1". Level2 stores verbs that are relevant in the laboratory domain. For example, level2-node1 (l2n1) might refer to the verb "measure" (and its synonyms in this domain). The third level consists of interrogatives such as "how", "what" and "why". Interrogatives help the NLQA system ascertain what type of information the user is seeking. "How" questions deal with procedures, "what" queries seek facts, "why" requires analysis, etc. Level4 stores keywords that are associated with difficulties. Words like "wrong", "error", "problem" and their synonyms are found here. These words indicate that the user is encountering a problem or difficulty and that the TA is being asked to provide some clarification. The fifth level contains property nouns such as "current", "voltage", "power", etc. The other levels incorporate concepts, which can be classified as falling under the same category[2]. Each level has a "default" node as its last node. This node does not store any of the keywords and is denoted as "l*df", where '*' is the corresponding level number (e.g. "l1df" for level 1). The occurrence of this node in a condition of a rule ensures that none of the words in the corresponding level are present in the user input, for this rule to be true.

Rule-bases

The NLQA framework discussed here is rule-based. The rules provide a mechanism for deciding what the user wants. One or more of these rules fire if their antecedents are satisfied. In this framework, the antecedents are the keywords provided by the user. The system response is in the form of a tutorial (audio/video/animation), a comment or a dialogue[3]. The antecedents are the level-node addresses of the words in the user input. The coding of keywords in this duplet fashion removes the need to re-write rules to account for synonyms. For a rule to fire, all of its "conditions" have to be met. These conditions are composed of options. A "condition" is satisfied if any of its constituents "option" is true. If the rule fires, the "result" stores the necessary action information that has to be output. XML has been used to encode these rules in a "rule base". A sample rule follows.

```
<rule no="1">
  <!-- how do I use/work with a dc
     supply? -->
  <condition no="1">
      <option no="1">l1n4</option>
  </condition>
  <condition no="2">
      <option no="1">l2n4</option>
      <option no="2">l2n9</option>
  </condition>
  <condition no="3">
      <option no="1">l3n1</option>
      <option no="2">l3n2</option>
      <option no="3">l3df</option>
  </condition>
  <condition no="4">
      <option no="1">l4df</option>
  </condition>
  <result>DCS_00: Tutorial to use a
      DC Power Supply.</result>
</rule>
```

The above rule reads as follows: If "l1n4" is present and if either "l2n4" or "l2n9" is present and either "l3n1" or "l3n2" or "l3df" is present and "l4df" is present in the user input encoded form then, output "DCS_00: Tutorial to use a DC Power Supply." as the result.

[2] For example, level7 represents a collection of technical terms in EE - "ohms", "kirchoff", "thevenin", "norton", "superposition", etc.

[3] If the query from the user requires analysis, the NLQA system calls the circuit recognizer, an algorithm written to determine if the student-connected circuit represents accurately the schematic in the laboratory experiment. The circuit recognizer then initiates a dialogue with the student to assist him/her in isolating the source of the difficulty. This part of the virtual TA is not described in this paper.

Knowledge Hierarchy

There is a rule base for every experiment (or module) in UVL[4]. Each of these files has rules that deal with concepts associated with the corresponding experiment. This division of knowledge based on individual experiments eases the task of rule building by making the designer of the system concentrate on the finer details at hand. Each of these rule bases has to be inter-related so as to maintain an application continuum. For example, the knowledge contained in a later experiment could require the basic knowledge in an earlier experiment ("prereqs" tag). This calls for creation of a hierarchy of rule bases. The following is an XML representation of a portion of this hierarchy.

```
<prereqall expno=" PrereqAll.xml" />
<exp no="7" name="Experiment 7">
    <kb> KBExp7.xml</kb>
    <totalprereqs no="2" />
    <prereqs expno=" KBExp2.xml" />
    <prereqs expno=" KBExp3.xml" />
 </exp>
<exp no="8" name="Experiment 8">
    <kb> KBExp8.xml</kb>
    <totalprereqs no="1" />
    <prereqs expno=" KBExp2.xml" />
</exp>
```

The tag "prereqall", points to the rule base (e.g. "PrereqAll.xml") that has the most basic rules about the application (UVL) itself. This rule base has rules to answer queries like "what is voltage?" or "what is a breadboard?" which are pre-requisites for understanding concepts in any experiment. Assuming that the user is performing "Experiment 7" in UVL, the following action would be inferred by the NLQA system (with respect to the above XML code): "Scan rule bases KBExp7.xml, KBExp2.xml, KBExp3.xml and PrereqAll.xml in that sequence for a matching rule."

Processing the User Sentence

The user input is first pre-processed by removing stop-words and punctuations. Some of the common words like, "and", "a", "an", "the", "or", etc. and almost all of the punctuations are stored in a file and referred to during this stage. The next stage involves traversing the vocabulary file to find if the user has used any of the keywords in the vocabulary file. If and when a keyword is detected in the user input its level-node configuration is noted. The following shows the word-by-word translation of the user query to the intermediate encoded form,

User Question: "How do I measure the voltage across a resistor?"

After Pre-processing the user query: "how do measure voltage across resistor"

Translation: "l1n1 (resistor) l2n1 (measure) l2n10 (do) l3n1 (how) l4df (no words from level4) l5n1 (voltage) l6df (no words from level6) l7df (no words from level7) l8df (no words from level8) l9df (no words from level9) l10n4 (volt) l11df (no words from level11)".

The order in which the translation occurs (l1n1, then l2n1, etc.) depends on the way in which the keywords have been arranged in the vocabulary file ("resistor" is at level 1 and hence is encountered first in the vocabulary file during scanning, "measure" is at level2, below level 1 in the file etc.). The presence of default nodes (like l4df, l6df, etc.) denotes that none of the keywords in all the nodes in the corresponding levels were present in the user input. Once the translation is done, the knowledge hierarchy is accessed, to find the related rule base files based on the students' current experiment. The current experiment information is obtained from the authoring environment (AE)[5]. If any of the rules fire, the corresponding results are passed on to the AE for further action.

Failure Scenarios and Recovery Strategies

What if none of the rules in any of the rule bases was satisfied? This could have been due to the following reasons,

1. Although the user input was valid and the system had a valid response in the form of a tutorial/comment/dialogue, the system could not relate some of the words in the user input with the keywords already present in the vocabulary file. Consequently, proper translation might not have been achieved to trigger a relevant rule. To resolve this, each word in the user query is fed to WordNet®[6] (Miller et al. 1998) for a list of synonyms. Each word in this synonym list would be examined in the vocabulary file. If any of the synonyms from WordNet® matched any of the keywords in the nodes in the vocabulary file, that particular word used by the user would be added as a new synonym under the matching node in the file. This process would be done for all the words in the user sentence. Now, due to the new additions to the vocabulary file, the system would recognize and translate the words in the user input to the appropriate level-node address to trigger a rule.
2. The second failure scenario can occur when the user query is valid, but insufficient to recognize a particular result. Such a situation would occur when the user

[4] Presently, there are twenty experiments that students can work on in the UVL.

[5] The interactive multimedia program might be thought of as a graphical user interface (GUI) between the NLQA system and the user. For brevity it is called the authoring environment (AE). The environment calls the NLQA system with the input query and gets in return the results of the rules fired.

[6] WordNet® is an online lexical reference system whose design is inspired by current psycholinguistic theories of human lexical memory.
(Source: http://www.cogsci.princeton.edu/~wn/)

asks a generic question, while the rule bases have been written to deal only with more specific cases of the same user query. In this case, the system needs to be able to pull out all the possible results that could help the user find a solution. This is done by retrieving all the rules, which contain in them non-default level node pairs from the encoded input. For example,

User question: "how do I measure voltage?"

Translation: "l1df (no keyword from level 1) l2n1 (measure) l2n10 (do) l3n1 (how) l4df (no keyword) l5n1 (voltage) l6df (no keyword) l7df (no keyword) l8df (no keyword) l9df (no keyword) l10n4 (volt) l11df (no keyword)".

Although the query seems straightforward, as there are no rules to deal with "voltage measurement" in general in any of the rule bases, it would fail to give a result in its first attempt. For any of the rules to fire, the system has to know either of the following,

- What to measure the voltage with (multimeter/oscilloscope)? (Or)
- For what component to measure the voltage across (resistor/potentiometer)?

This is because, the related rules in the rule bases deal with specifics of measuring voltage. The tutor is trying to determine if it should show how to measure the voltage with a specific instrument or if it should explain how to measure voltage across an element. As the query does not produce a result, the system would assume "failure scenario 1" and try to call WordNet® for all the words it does not know. But as all the words in the above sentence are already known ones, effectively there are no new words to learn. Now, as the second failure scenario is assumed, the default level-nodes would be removed from the encoded form resulting in "l2n1 l2n10 l3n1 l5n1 l10n4" (measure-do-how-voltage-volt). Using this new encoded form, the system checks all the rule bases in the knowledge hierarchy for rule matches. Unlike before, even if only one of the five level-node pairs were present in any rule, the rule is selected as a possible candidate.

Each level-node (or keyword) would have its own importance in deciding how close the user query is related to the extracted rule. "voltage" or "measure" has a greater weight assigned and conveys more than "how" to the system. The rules with the maximum net weight would be the most preferred rule to answer the query. The authoring environment can decide the number of such rules to display for the user to select from and would communicate it to the NLQA system through the parameter file that would store the weights (node priority file, which would also be XML based). The output from the program for the "how do I measure voltage" input would be,

REPHRASE

OS_03: Tutorial for oscilloscope (Use oscilloscope to measure ac voltage).

DMM_01: Tutorial to use DMM (measure voltage).

OS_02: Tutorial for oscilloscope (Use oscilloscope to measure dc voltage).

Comment: Instantaneous voltage is the voltage at any particular instant of time - $V(t) = Vm * \sin(\theta)$.

Comment: Note the maximum height of the wave is its peak amplitude or peak voltage. You can measure this using the oscilloscope.

The results have been arranged in the decreasing order of the net weight of the rule. If the rules have the same weight, the rules from the earlier experiments follow those of the later ones (as described in the knowledge hierarchy) in the result listing. The user has an option to choose from any of the ordered results or can rephrase the query. The missed query along with the results is stored in a history file to improve the system response in future versions.

3. The third and last failure case occurs when the system does not have the knowledge required to answer the user query. In this case, the system records the user query to a file (for analysis and future improvements of the rule bases and vocabulary file). The system would prompt the user to re-enter his/her query in a clear and complete manner.

Knowledge Acquisition

The greatest drawback of any rule-based system is the laborious process of writing rules. To ease this process and to enable a rapid building of NLQA applications, we discuss briefly the knowledge acquisition tool (KAT) in this section.

Introduction to KAT

The knowledge acquisition tool (KAT) can be easily installed by the domain expert to build all the domain specific parts (vocabulary, rule bases, knowledge hierarchy, stop-words and the node priority files) of the NLQA system. No extra technical expertise, other than the common GUI experience and a basic knowledge of how rules are fired is required to use the tool.

The tool prompts the expert to enter queries anticipated from the end-user, recognizes words from these queries and builds rules. The tool also helps the expert identify keywords while building rules in this process. In this way the tool simultaneously builds the rules as well as the domain vocabulary and synchronizes them. Apart from this, the tool has features to add stop-words, to create rule base hierarchies and to assign weights to keywords. It also has features to optimize the order of rules in rule bases, by placing the more specific rules on top of the file and moving the general rules down, thereby making sure that the general rules do not fire when there exist more specific ones.

KAT is a project-based environment. To build a NLQA system for a new domain, the expert creates a new project and creates/attaches the domain specific files to it. Figure 2 shows the project environment in KAT.

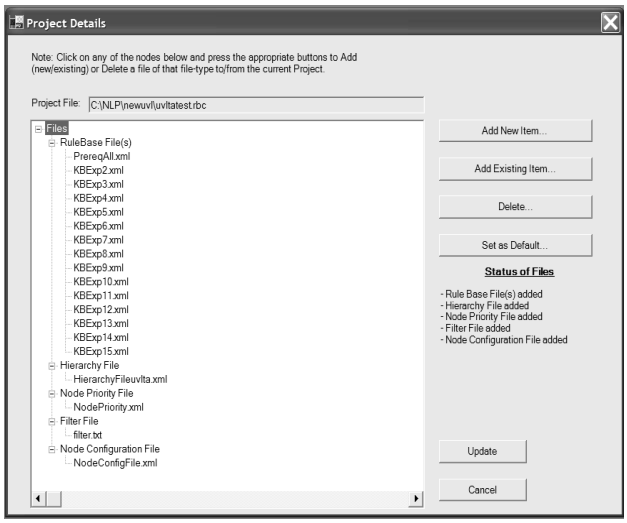

Figure 2: KAT Project Environment

The 'Rule base File(s)' shows all the rule bases in the application. The 'Node Configuration File' refers to the vocabulary of the domain, the 'Hierarchy file' to the knowledge hierarchy, the 'Filter File' to the stop-words file and the 'Node Priority File' to the weighting mechanism of keywords. Each of these files can be either created from scratch or extracted from some other existing project.

Building Domain Vocabulary and Rules

Figure 3 shows the screen for rule building in KAT. The domain expert enters the anticipated user input and the corresponding result ID (the tutorial/comment/dialogue files). Words parsed from the sentence are listed for the expert to classify as 'necessary' or 'optional' keywords. Necessary keywords are those that have to be present in the user query for the rule to trigger. Optional keywords need not be present in the query, but help in weighting and listing the rule favorably (to be displayed to the user for selection by the AE) if none of the rules trigger and "failure scenario 2" is encountered. Other equivalent anticipated user queries (having the same result ID) could be entered; their keywords extracted and combined with already identified keywords into a single rule using the 'New Sentence' button.

When the expert clicks on 'Create Rule', the system accesses the domain vocabulary to get the level-node code corresponding to the keywords and formulates the rule in the encoded format. If the words in the sentence are not already present in the domain vocabulary but have been identified by the expert as necessary or optional keywords, the screen for adding these words as new domain keywords appears (Figure 4). Thus, using KAT links vocabulary building with the rule building process.

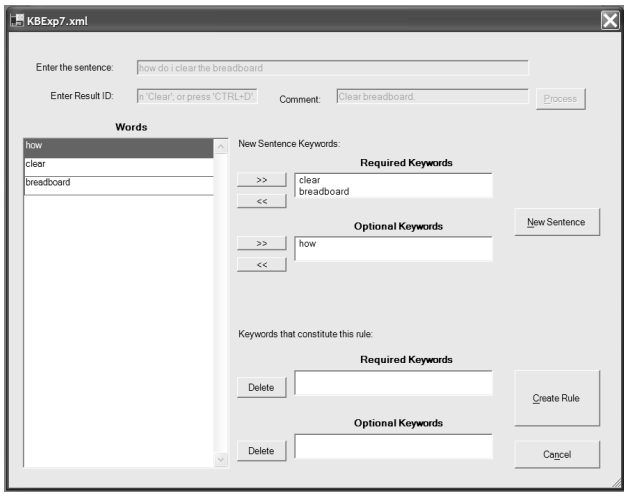

Figure 3: Rule-building in KAT

The expert also has the freedom to add keywords as he/she pleases directly (not while rule building), using the form shown in Figure 4. New levels, nodes and synonyms can be added using this form. This vocabulary editor is very useful in the initial stages of the project when there are many identified keywords to be entered into the system.

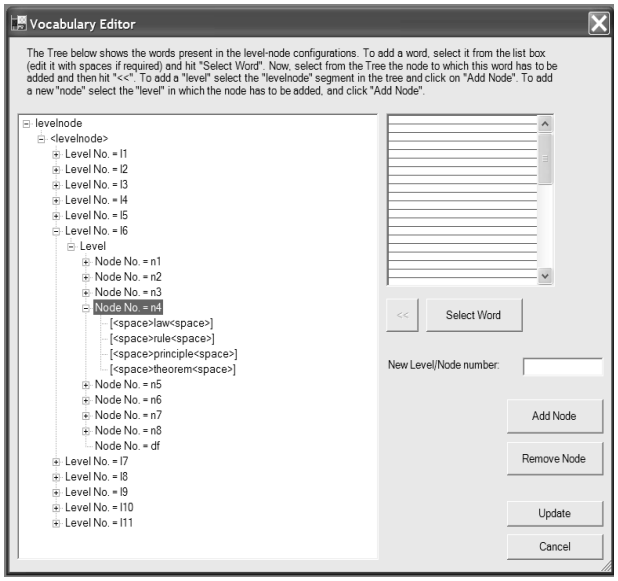

Figure 4: Vocabulary Editor in KAT

KAT also features a form for modifying rules in a rule base. Rules can be viewed, added, deleted or modified using that form.

Organizing the Knowledge Hierarchy

Figure 5 shows the form that is used to build the knowledge hierarchy.

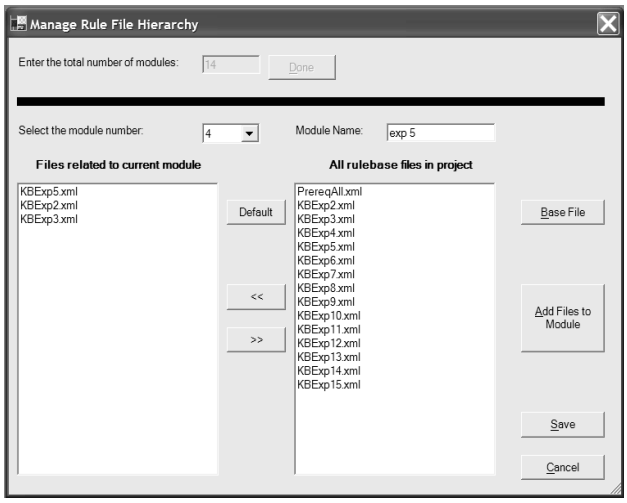

Figure 5: Knowledge Hierarchy Editor in KAT

The expert specifies the total number of modules (experiments, in case of UVL) in the domain. Each of these modules has a rule base directly associated with it. Figure 5 shows an example, where the total number of modules is 14 (excluding PrereqAll.xml). For the fourth module, the primary rule base (set by selecting a file from 'Files related to current module' and then clicking the 'Default' button) is KBExp5. KBExp2 and KBExp3 are prerequisite rule bases for module 4. The prerequisite for all the rule bases can be set by selecting a file from the 'All rule base files in project' list and clicking the 'Base File' button. Clicking 'Add Files to Module' associates the current module files to the respective modules. 'Save' writes the knowledge hierarchy to the XML file specified by the expert (as 'Hierarchy File' in project settings).

Stop-words and Keyword Priorities

KAT permits the expert to specify stop-words that occur often in the expected user queries, but add little or no meaning with respect to QA in the domain. Removing such words speeds-up the system response and eliminates the need to call WordNet® unnecessarily.

KAT also has a node priority editor screen. Each level as discussed before is assigned a weight. This weight indicates the importance of the level for the domain. The expert can assign these weights by selecting the appropriate level and entering a positive number between 1 and 10 (10 refers to the most important level in the domain) as deemed fit.

Enhancements and Future Ideas

We have implemented the framework in other domains (healthcare). We plan to incorporate spell-check mechanisms in the future versions of the framework to make the system resilient to typographical errors. Automatic learning of words from WordNet sometime leads to misinterpretation of rules due to the inappropriate context of the word as it applies to UVL. This problem is under study. The possibility of linking such domains to achieve a cross-domain NLQA is also being investigated. A dialogue system using the framework is also being considered.

Conclusion

Basic string matching or statistical analysis techniques alone would not result in a good NLQA system. Context is very important when dealing with language understanding. To assess the context, various heuristics or expert system techniques should be employed into natural language understanding systems. The framework that we have presented addresses this by viewing NLQA as an expert system problem. Understanding NL questions from domain keywords is a very effective way to cater to specific user requests, accurately. Knowledge acquisition tools, like the one discussed here, can be very useful in building such systems rapidly.

Acknowledgments

Partial support for this work was provided by the National Science Foundation's Division of Human Resource Development through grant HRD #0004292.

References

CDC 1995. "National Health Interview Survey." Center for Disease Control. Hyattsville, MD, USA. 31-32.

NSF 1997. "SBIR Phase II: Computer Simulation of Science and Technical Laboratory Exercises for Physically-Disabled Students."

NCES 1996. "The 1996 National Postsecondary Student Aid Study." U.S. Department of Education. Washington, DC, USA. 72-73.

Duarte, M. and Butz, B.P. 2002. "An Intelligent Universal Virtual Laboratory (UVL)", 34th Southeastern Symposium on System Theory, Huntsville, MD, USA.

Weizenbaum, J. 1996. ELIZA - A computer program for the study of natural language communication between man and machine. Comm. ACM 9:36-45

Whalen, T. and Patrick, A. 1989. Conversational hypertext: Information access through natural language dialogues with computers. In *Proceedings of the Conference on Human Factors in Computing Systems: CHI '89*, Austin, Texas, USA, April 30-May 4, 1989, p.289-292, ACM Press.

Miller et al. 1998. WordNet: An Electronic Lexical Database; MIT Press.