A QUESTION ANSWERING SYSTEM FOR PROJECT MANAGEMENT APPLICATIONS

Jinxing Cheng[1], Bimal Kumar[2] and Kincho H. Law[3]

ABSTRACT

The usage of computer applications in the construction industry has steadily increased over the years, as has the complexity of many software applications. It is difficult for project personnel to become familiar with these ever-increasingly complex tools. Furthermore, the causes of many practical problems, such as project delays or escalating costs, are often not obvious from the outputs of these tools. A question answering system can potentially provide a means to directly extracting answers from these computer outputs. This paper examines various issues involved in building such a question answering system. In particular, emerging industry standards, such as ifcXML, are adopted as the knowledge representation format, and thus alleviate the manual effort to build a knowledge base. We explore the mechanisms of utilizing information in the knowledge base for question understanding. A prototype question answering system has been built and tested to illustrate the potential usefulness of such a system for project management applications.

KEYWORDS

Question Answering, XQuery, ifcXML, Knowledge Representation

[1] PhD Student, Civil and Environmental Engineering Department, Stanford University, Stanford, CA 94305-4020, email: cjx@stanford.edu

[2] Professor, School of the Built and Natural Environment, Glasgow Caledonian University, UK, email: b.kumar@gcal.ac.uk

[3] Professor, Civil and Environmental Engineering Department, Stanford University, Stanford, CA 94305-4020, email: law@cive.stanford.edu

1. INTRODUCTION

Natural language processing technologies have been used in many applications. Examples include database access (Woods 1973, Pereira 1983), machine translation (Vasconcellos and Leon 1985), data extraction from text (Jacobs and Rau 1990), information retrieval (Baeza-Yates and Ribeiro-Neto 1999), and text categorization/summarization (Hovy and Lin 1999). In the case of construction projects, one can think of applications for almost all of these areas.

In this paper, we focus on the use of NLP technologies to help answer questions based on semi-structured data generated from project management tools. This application is closely related to research in data extraction from text and database access. A combination of information retrieval and NLP technologies can provide a powerful tool in all sorts of ways. For example, a project manager may use various software tools for scheduling, cost estimating, and reporting purposes. However, to discover the reasons why certain activities delay a project or escalate the project costs, human expertise and efforts are still needed to manually examine the outputs from these tools. Using NLP technologies it may be possible to convert natural questions into query expressions, so that such information can be obtained from the software outputs. Data from various tools can be extracted, converted into structured or semi-structured formats, and even stored into a database. The query results on semi-structured data will be used to generate answers (again using NLP) which may not have been obvious from the initial outputs of the software programs.

The main objective of this work is to develop a framework for an NLP based system for extracting useful information from semi-structured pieces of text. This framework is implemented as a prototype system containing knowledge and information from the domain of construction project management. Marked up using domain specific ontology standards, such as ifcXML (Liebich 2001), the semi-structured texts consist of the outputs from various computer applications used in a construction project. It is hoped that the prototype system will be able to extract useful information from these semi-structured texts, which would otherwise not be obvious or possible to obtain directly from these applications. The framework developed in this system is being implemented as a QA (question answering) system that can assist project personnel in making inferences about the project, based on information obtained from various project management tools.

There have been many efforts in developing question answering systems. For instance, efforts have been made to develop natural language interfaces to databases. Androutsopoulos et al. (1995) discussed various methods and solutions available in translating natural questions to database queries. Although some solutions seem promising in a narrowly defined domain, it is difficult to apply these technologies to construction projects, where database interfaces are not typically supported by the application software. As another example, Callison-Burch and Shilane (2000) developed a question answering system to query information about a family tree. This system used the Knowledge Interchange Format (KIF) (Genesereth and Fikes 1992) files as the knowledge representation system, and used a Java-based Theorem Prover (JTP) (Frank 1999) to infer answers. To develop such a system, a knowledge base needs to be created manually, which makes it difficult to generalize the system to other domain areas. Even within the same domain, this system may not scale well over a number of individual projects, since a knowledge base needs to be created manually for each project. Zajac (2001) used a more general ontology-based semantic approach for question answering. Both the questions and source texts are parsed into semantic expressions. However, this approach assumes that the source texts are expressed in natural sentences, from which semantic information can be extracted; thus it cannot be directly applied into the project management domain, where information is stored in various internal formats, usually either in plain text files or in semi-structured data files. Our work aims to develop a question answering system, which is scalable over different projects and considers the characteristics of construction project information. Specifically, we plan to take advantage of current development of query languages for XML data and the industry-based ontology standards.

The main contribution of this work is in the area of using non-document based retrieval of information by combining information from multiple sources, which is mainly accomplished by utilizing domain-specific ontologies in ifcXML. The fact that we use NLP techniques to convert questions into formal pattern matching language means that our work also has important implications for User Interfaces in engineering domain software. Most of the effort in NLP so far has been in the area of interacting with documents in natural sentences; this work, however, takes into account domain-specific issues, which makes it much more relevant and effective (Diekema et al. 2003). The lessons and experiences gained from work on domain-independent fact based questions do not necessarily ensure an effective QA system in specialized domains (Diekema et al. 2000) such as project management.

This paper is organized as follows. Section 2 briefly reviews related technologies, such as IFC/ifcXML, XML query languages and engines, POS tagging and chart parsing tools, and WordNet, which are employed to develop the question answering system. Section 3 discusses the issues of knowledge representation and organization. Section 4 describes the process of parsing and understanding questions; we discuss in detail how to utilize the ifcXML schema and existing ifcXML files in the knowledge base to help understand questions. In Sections 5 and 6, we briefly discuss how to search for answers in the knowledge base and how to generate answers, respectively. Section 7 describes the prototype framework and system implementation. An example demonstration of the system is presented in Section 8. Finally, Section 9 summarizes the status of the current development and discusses future work.

2. RELATED TECHNOLOGIES

Many recent and current developments, such as ontology standards, query languages, language parsers and information retrieval, can be employed to build a question answering system. In this work, various technologies are used, including XQuery (W3C 2001), GNU Kawa's Qexo (Brothner 1998), a POS tagging tool (Schröder 2002), a chart parsing tool (Klein and Manning 2001a) and WordNet (Fellbaum and Miller 1998). The following briefly describe these related technologies.

2.1. IFC AND IFCXML

IFC (Industry Foundation Classes) (IAI 1997) is a data representation standard for defining product data for architectural and construction applications. Recently, IFC has also been extended to support data exchange for cost estimation and project management (Froese et al. 1999). In brief, based on the EXPRESS language, IFC is designed to exchange data among Architecture, Engineering, Construction and Facilities Management (AEC/FM) applications.

XML (Extensible Markup Language) is a meta-markup language that consists of a set of rules for creating semantic tags used to describe data (Young 2001). XML provides a mechanism to describe an object as a hierarchy of elements. Due to the popularity of XML, many efforts have been invested in proposing XML schemas as ontology standards in the construction and manufacturing industry as well as for business applications. An effort has been made to translate IFC/EXPRESS source files into XML format, called ifcXML (Liebich 2001). The translation takes place in two steps: First, the IFC/EXPRESS source is translated into the raw ifcXML schema, which enables the exchange of IFC data in XML formats. The raw schema is further optimized into more native XML schema. In short, ifcXML not only deals with geometry and product data, but also supports life-cycle project information, including architecture, HVAC, construction and facility management (Liebich 2001).

2.2. XML QUERY LANGUAGE

XML query languages are designed to query information from XML files. Many XML query languages are available, such as XQL (Robie 1999), XML-QL (W3C 1998), LOREL (Abiteboul et al. 1997), Xpath (W3C 1999), and XQuery (W3C 2001). The query capability of XQL is limited at its current state, while the research on LOREL is no longer active. In contrast, XQuery, based on XML-QL and Xpath, is a full-featured query language and is emerging as a standard. Our initial investigation shows that XQuery is an appropriate language for our prototype implementation of a question answering system.

XQuery (W3C 2001) is an XML query language jointly defined by the XML Query Working Group and the XSL Working Group and is designed to be applicable to all types of XML data sources. XQuery uses a syntax similar to SQL. For example, the following XQuery sentence can be utilized to query the start date of the activity *ELPV* (Eng Layout & Physical Ver_n):

```
for  $ws in document("tuto.xml")//WorkSchedule
where    $ws/@identifier = "ELPV"
return   $ws/@startTime
```

2.3. XML QUERY ENGINE

Although XQuery is still in its W3C Working Draft version, many vendors have implemented XQuery, such as Xquench (SourceForge 2002), XQEngine (FatDog 2002), Galax (Siméon 2001), and GNU Kawa Qexo (Brothner 1998). In this work, we employ the GNU Kawa Qexo as the query engine because of its high performance and easy usage, as well as its open source.

Qexo is an open source project and a partial implementation of the XML Query language from GNU Kawa (Brothner 1998). There are two ways to use Qexo. Qexo can be used in an interactive environment, in which users can input query sentences at the command line. In addition, Qexo can also compile XQuery sentences into Java byte codes, which significantly improves the query efficiency. In this work, the latter approach is adopted; XQuery expressions are compiled into Java byte codes to automate the process of answer searching.

2.4. POS TAGGING AND CHART PARSING TOOLS

Part-Of-Speech (POS) tagging is designed to label each word in a sentence with its appropriate part of speech. For instance, the word “finish” in the sentence, “When did the project finish?” should be labeled as a verb. There are many POS tagging tools available. This work employs ICOPOST (Schröder 2002), which is a set of free POS taggers written in the C language.

A grammar parser is used to recognize the structure and organization of words in a sentence. Among many available grammar parsers is a context-free grammar parser, which assumes that phrases with the same part of speech can be interchangeable regardless of the specific context. In this research, we have used the context-free grammar chart parser developed at Stanford University (Klein and Manning 2001a). The chart parser takes a grammar file and a lexicon file. All possible parses, together with the best parses based on the probability analysis, will be output as the parsing results.

2.5. WORDNET

Developed by the Cognitive Science Laboratory at Princeton University (Fellbaum and Miller 1998), WordNet is a lexical reference system, in which words are organized into synonym sets. WordNet can be used online; it can also be installed and used on different platforms, including Microsoft Windows and Unix environments. WordNet can help a question answering system to identify synonyms. For example, verbs “start” and “begin” will be recognized as synonyms by WordNet. The synonym information can be used to help match a question with an appropriate rule, as we will discuss in Section 4.4.

3. KNOWLEDGE REPRESENTATION AND ORGANIZATION

3.1. KNOWLEDGE REPRESENTATION USING IFCXML

Knowledge representation is crucial in building a question answering system. Ideally, the knowledge base should be automatically built, based on the existing information in the domain. In the project management domain, project information in various applications is usually stored in different internal formats. In previous work, wrappers have been built to retrieve project information from various sources and convert the information into standard formats (Cheng and Law 2002, Cheng et al. 2002). Many ontology standards exist in the A/E/C domain, such as STEP and IFC/ifcXML. These ontology standards provide standard terms and, often, relationships among the terms. In this research, we use ifcXML as the knowledge representation format. Project information from various applications can be extracted and translated into ifcXML files.

In the prototype application, we focus on the project management domain; thus, only a small portion of the ifcXML schema is used. Specifically, in the ifcXML schema, the *WorkSchedule* element holds the overall scheduling information, such as the start time and duration, while the *ScheduleTimeControl* element holds further descriptions of scheduling information, such as *actualStart*, *earlyStart*, *lateStart* and *scheduleStart*. The *RelSequence* element, on the other hand, is used to express the dependency relationships among activities.

3.2. ADVANTAGES AND LIMITATIONS OF IFCXML AS KNOWLEDGE REPRESENTATION FORMAT

IfcXML is emerging as an industry standard and has many advantages for being adopted as a knowledge representation format. First, ifcXML provides many of the terms and relationships commonly used in project management applications (Liebich 2001). In addition, ifcXML provides XML-based schemas, which are easy for querying and transferring on the Internet. Second, as discussed earlier, there are many existing tools that can be used to parse and query XML data. Finally, ifcXML has the power to model data from various project management applications, which include not only product data but also process and activity information.

The ifcXML schema is translated from the IFC/EXPRESS source (Liebich 2001). IFC was initially designed for exchanging graphic data among CAD applications. The initial intent clearly was capturing information related to product models. However, many efforts have recently been invested in extending IFC to model project information throughout the entire building life-cycle. In particular, IFC 3.0 will significantly extend the coverage of the models in construction project management (IAI 2002).

One concern in using ifcXML is that the translation of IFC to ifcXML is not a one-to-one mapping. Information can be lost during the translation process. The information loss, however, is insignificant and does not hinder the objective for research prototyping. Furthermore, the system can easily accommodate new terms and relationships as they become available in the ifcXML schema.

3.3. KNOWLEDGE ORGANIZATION

As the size of a knowledge base grows, it is necessary to partition the knowledge base into smaller chunks. These chucks are called knowledge modules, each of which addresses a sub-problem of the overall problem domain (Dym and Levitt 1991). Generally, the main issues that need to be considered when organizing a knowledge base are:

- *What* needs to be represented in the knowledge base? This issue is related to the *content* of the knowledge base.
- *How* should we represent the content that needs to be represented? That is, an approach must be selected to organize the knowledge, and *formalisms* (like rules, frames, semantic nets, objects or a combination of these etc.) should be used.

For project management applications, the knowledge base can be organized as a set of knowledge modules, each representing a sub-domain, such as schedule, cost, organizational model, etc. Meta-knowledge (i.e. knowledge about knowledge) is then defined to guide the processing of the facts encoded in the knowledge base; in other words, the meta-knowledge can help a question answering engine search for relevant information. In this work, the meta-knowledge is encoded as sets of patterns, which serve as the indices to different knowledge modules. When a question is posed, these *patterns* are used to identify the relevant knowledge modules from which to retrieve the answer. Figure 1 shows a schematic representation of the knowledge base. A project contains information from several sub-disciplines, each of which usually comes from a corresponding project management application. Moreover, within a sub-discipline, information about different stages of the project can be stored in the knowledge base.

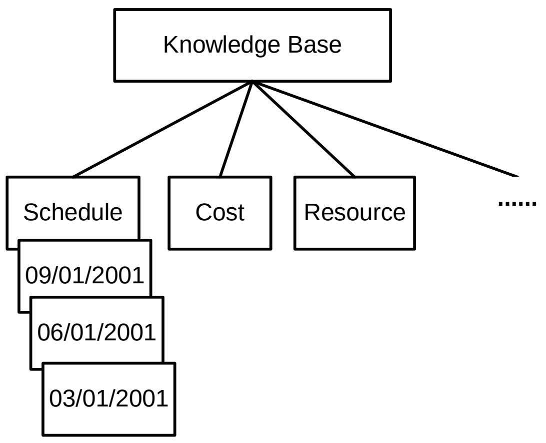


Figure 1: A Context Tree For Knowledge Organization

4. PARSING AND UNDERSTANDING NATURAL QUESTIONS

Understanding natural questions is rather difficult for computer applications, partly due to the fact that there can be too many variations of natural language questions. Even if we limit our research to a small predefined domain, for instance, the project scheduling domain, there are still significant variations of possible questions, such as:

1. *When did the PouringConcrete activity start?*
2. *Why should the PouringConcrete activity start before ErectingBeams?*
3. *Which subcontractor submitted scheduling changes yesterday?*

Although all these questions are (1) syntactically correct; (2) semantically sound; and (3) within the project scheduling domain, not all of these questions can be answered from the knowledge base. For example, we may not have information for questions 2 and 3 in our ifcXML files. For a practical question answering system, questions from users can be either syntactically incorrect, semantically unsound, or both.

4.1. ANALYZING IFCXML TREES

The ifcXML schema can be utilized to facilitate the understanding of questions. Using the information in the schema, we can predict what kind of questions we can answer. We can safely ignore the questions with no answers in the knowledge base; in other words, we do not need to understand the exact meanings of many questions that we cannot answer from the existing knowledge. A preliminary analysis of such questions may be enough to discard them. For example, for the question *"How many governors of California have been democrats during the past 50 years?"* after a preliminary analysis, we know that no rule in the knowledge base matches this question; thus, there is no need to further analyze the exact meaning of the question.

To utilize the ifcXML schema, the first step is to analyze the tree structures of ifcXML files. We have developed a Java program, which can analyze all the elements, attributes, and relationships in the ifcXML files. Based on the analysis, we can predict possible questions that we can answer and express the questions as rules. Each rule contains one relation word and several parameters, such as the rules *(duration activity)* and *(finish activity).* The rule *(duration activity)* means that an activity lasts a specific number of days, while the rule *(finish activity)* specifies the finish date of the activity. Usually, the first word in the rule specifies the relation, while the remaining words represent the parameters of the rule. Figure 2 shows part of the tree structure of the ifcXML schema about project scheduling. The leaf nodes are attributes, while all other nodes are elements.

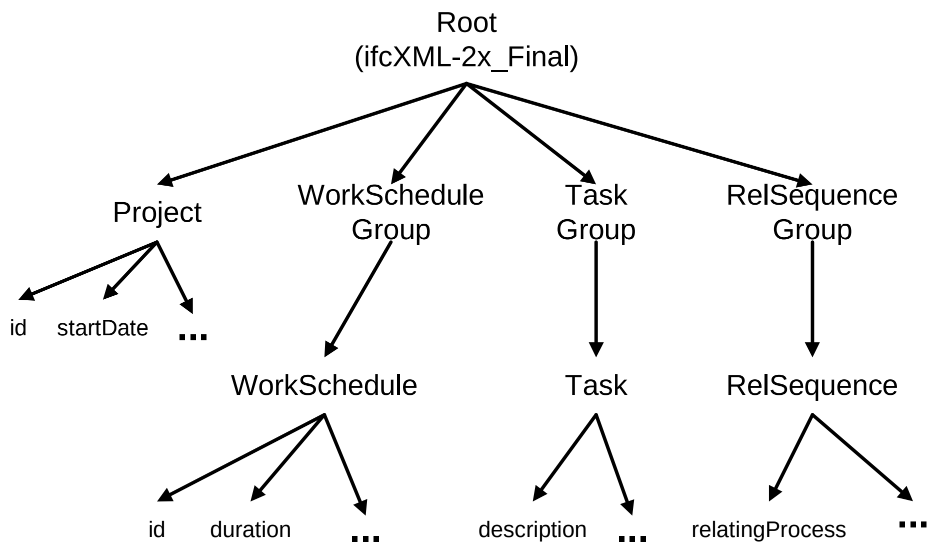


Figure 2: Tree Structure of IfcXML Files

According to the tree structure, there are several types of questions that the system should be able to answer, for example:

- Questions inquiring the attribute value of an element
- Questions asking which element has certain attribute value
- Questions involving several elements in the ifcXML tree structures

The first two types of questions are relatively straightforward. Using a leaf node (attribute) and its parent node (element), we can produce rules for possible questions of the first type. For instance, after analyzing tree structures of the ifcXML schema, we can automatically generate rules, such as *(startDate Project)* and *(duration WorkShedule)*. The rule *(startDate Project)* means that we can answer questions such as "What is the start date of the project?" However, this automatic analysis is not perfect. For example, in ifcXML, we use the element *WorkSchedule* to represent the schedule information for an activity; in practice, however, people usually ask about the duration of an *activity* instead of a *WorkSchedule*. That is, we should use *(duration activity)* instead of *(duration WorkSchedule).*

The third type of questions, on the other hand, is quite complex. We first need to determine the relationships among different elements. Based on the relationships, we then need to predict possible questions that we can answer and generate the corresponding rules for the questions. Again, these automatically generated rules need to be examined manually by experts. One sample rule is *(description (hasDuration number))*. For this rule, the possible question is to inquire the description of an activity with a specific duration. The *(hasDuration number)* will return an activity name, which will then be used to obtain the activity description. The manual work involved in generating rules is worth the effort. Since ifcXML files for different projects have the same XML tree structure, we only need to manually examine these rules once. These rules can then be used for a variety of construction projects.

4.2. TAGGING QUESTIONS

The Part-Of-Speech (POS) tagger is used to provide POS information for individual words. For example, using the POS tagger, we can find out whether a word is a noun or verb. The POS information can be used to help understand the meanings of words. In particular, it can help us identify whether or not a word is a potential relation word or a parameter in a rule.

Questions need to be processed before we can tag them. In particular, we need to separate the words and punctuation marks in the questions. The following examples show some sample tagging results:

When WRB will MD activity NN ID100 NN start NN ? .

How WRB many JJ successors NNS does VBZ activity NN ELPV NNP have VBP ?

When WRB will MD ELPV NNP end VB ? .

Here NN (NN, NNS or NNP) represents a noun, VB (VB, VBZ or VBP) represents a verb, WRB represents a wh-adverb, JJ represents an adjective, and MD represents a modal word. We can see that the POS tagger does not always provide the correct POS information. For example, it tags the word “start” in the first sentence as a noun, which is actually a verb. Nonetheless, the POS information provides a good basis for understanding questions.

4.3. PARSING QUESTIONS

We use a context-free grammar chart parser for parsing questions. To use the chart parser, we first need to create both a grammar file and a lexicon file. While it is possible to write a grammar file for questions in the project management domain, it is a tedious, if not an impossible task, to create a lexicon file for all possible questions, since the lexicon file must contain all the words which can appear in the questions.

We can extract a grammar file from the Penn Treebank, a human-annotated corpus consisting of over 4.5 million words of American English (Marcus et al. 1993). However, the extracted grammar file is huge, while most of the grammar rules are not useful for questions.

The size of grammar rules will significantly affect the parsing process. On the other hand, a much smaller grammar file is possible for questions within a small domain. As a result, a grammar file for questions in the project management domain is developed. Figure 3 shows part of the grammar file, which is largely based on a heuristic observation. As an example, the grammar rule "*S -> WRB MD NP VP ? %%0.1*" indicates that a question sentence can be rewritten into a wh-adverb, a modal verb, a noun phrase, and a verb phrase. The grammar rule "*NP -> NNS %%0.1*" indicates that a noun phrase can be rewritten as a singular noun, in which the number *0.1* represents the probability that a noun phrase will be rewritten as a singular noun. The probability values are not used by the context-free chart parser in this work; rather, these values are provided to conform to the required format, since other probabilistic context-free grammar parsers need these probability values.

```
S -> WRB MD NP VP ? %%0.1
S -> WP MD NP VP ? %%0.1
NP -> NN %%0.1
NP -> NNS %%0.1
NP -> NNP %%0.1
VP -> VB %%0.1
VP -> VBD %%0.1
……
```

Figure 3: Grammar File for the System

For the lexicon file, we can also extract lexicons from the Penn Treebank. Developed by the Stanford Natural Language Process Group, the standalone program ExtractPTBRules (Klein and Manning 2001b) can be used to extract lexicon files from a collection of the Penn Treebank sentences. However, even if we extract lexicons from a large collection of documents, many words in the questions may still not be included in the lexicon file; as a result, the chart parser will have difficulties in parsing the question.

A simple but effective approach is to utilize the POS tagging results. We can dynamically generate a lexicon file based on the tagging results. Obviously, all words in the question will appear in the lexicon file. Assuming that the question is syntactically correct, the chart parser will always find a parse for a question. Figure 4 illustrates this approach.

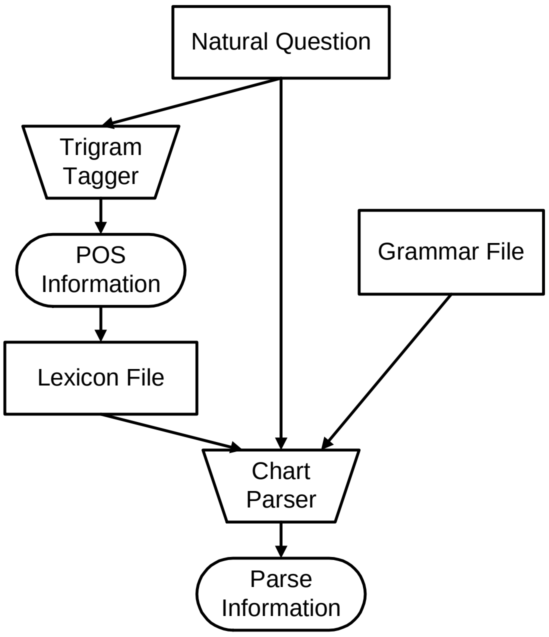


Figure 4: Tagging and Parsing Questions

Once the grammar and lexicon files are ready, we can use the chart parser to parse the questions. All possible parses, together with their corresponding probability values, will be generated. In addition, the best parse is also available. Figure 5 illustrates the process of generating a lexicon file for the question, while Figure 6 shows the parsing results of an example question.

```
Question:
      When did ELPV begin ?
POS:
      When WRB did VBD ELPV NNP begin VB ? .
Lexicon File:
      WRB -> When %%0.1
      VBD -> did %%0.1
      NN -> ELPV %%0.1
      VB -> begin %%0.1
      ? -> ? %%0.1
```

Figure 5: Preparing A Lexicon File

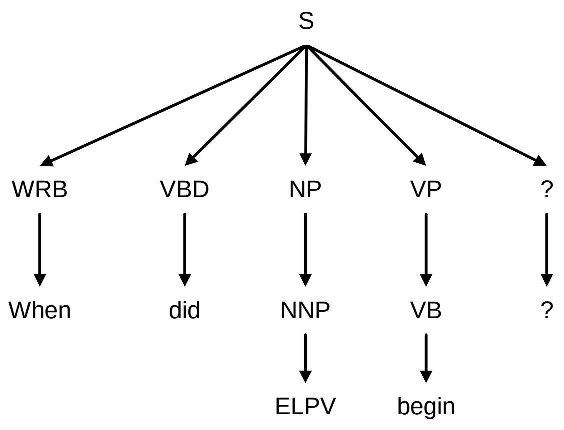


Figure 6: Parse Tree of A Sample Question

4.4. ANALYZING CONCEPTS AND MATCHING RULES

In this project, ifcXML is used as the knowledge representation format. Useful information can be obtained from ifcXML files before we actually start processing questions. For example, we can parse ifcXML files and store all activity names on a list. Later when the system encounters a question, it can search the list; if a word is found on the activity list, it is a strong indication that this word represents an activity name. Thus, this approach significantly helps the system understand questions.

Earlier, we discussed possible questions that we can answer from the knowledge base, and how to express them in rules. We also discussed how to obtain the POS and parsing information for the questions. Based on the information above, we can analyze the concepts in the questions and match the questions with corresponding rules. Usually, the most important words for understanding questions are the question words, the nouns and the verbs.

- Question words, such as *when*, *what*, *how* and *does*, determine the type of the question; in addition, these words also imply what kind of answers users expect.
- Nouns usually correspond to object names, such as activity names, in ifcXML files. Sometimes, nouns can also be used to express relations in the rules. For example, in the following question:

  What is the start date of Sim_Gates?

  The noun phrase "start date" corresponds to the relation "start" in the rule *(start activity).*
- Verbs usually imply the rule to which a question should be categorized. For instance, in the following sentence:
  When does the task Sim_Gates end?

The verb “end” is a strong indication that this sentence should be categorized into the rule *(end activity)*, while the word “Sim_Gates” corresponds to the *activity* parameter in the rule.

In addition, WordNet is used to categorize synonyms into the correct rules. For example, to ask the start date of an activity, we may use either *start* or *begin*. Using WordNet, we can categorize both situations into the rule *(start activity).*

Rule matching is based on probability scores. Based on the concept analysis, we can match a question with a set of possible rules. The probability scores will be calculated for each question and rule pair. The rule with the maximum probability score will be chosen as the correct rule to generate XQuery expressions if the maximum score exceeds the threshold value set in the system. Otherwise, no matching will be assumed by the system, which will lead to an answer like “No answer can be found from the current knowledge base.” In particular, the following information is used in calculating the probability scores:

- Relation words, which include the meaning and number of relation words. For example, the relation word “begin” in a question will match the word “start” in the rule *(start activity)*; thus, this rule will likely have a high probability score.
- Object names, which include the type and number of object names. The number of tasks and actors in a question is important in matching the question with possible rules.

5. SEARCHING FOR ANSWERS IN THE KNOWLEDGE BASE

To search for answers in the knowledge base, we need to translate questions into XQuery expressions, which can then be directly executed on the knowledge base. Translating questions into XQuery expressions is not a trivial task. Therefore, a Java program has been developed to analyze the tree structure of ifcXML files and generate rules for possible questions; meanwhile, the Java program also produces XQuery expressions for each rule. For example, the rule *(duration activity)* will be translated into the following XQuery expressions:

```
for  $ws in document("$xmlfile1")//WorkSchedule
where    $ws/@identifier = "$1"
return   $ws/@duration
```

There are two parameters in the XQuery expressions. The first parameter *$xmlfile1* appears in most rules; it represents one of the ifcXML files in the current project. The second parameter *$1* corresponds to the activity name in the rule.

When a question is parsed, it is categorized into a rule, based on the syntactic and semantic information of the question. As an example, the question “What is the duration of the task Sim_Gates?” will be categorized into the rule *(duration activity)*, where the value of the parameter *activity* is *Sim_Gates*. Suppose the corresponding ifcXML file in the current project is *p3_tuto.xml*, the following XQuery expressions will be generated for this question:

```
for  $ws in document("p3_tuto.xml")//WorkSchedule
where    $ws/@identifier = "Sim_Gates"
return   $ws/@duration
```

The XQuery expressions are then compiled into Java byte codes by the Qexo query engine. Finally, the generated codes are used to search the knowledge base for the answer.

6. ANSWER GENERATION

The answer generator first needs to parse the query results from the query engine. In addition, it needs to consider different types of questions. For instance, for a *wh-question*, a question with specific information is usually expected. For a *how many* question, on the other hand, we should give a specific number. In contrast, for a *yes* or *no* question, a yes or no answer is usually sufficient. In most cases, for a *wh-question*, if the XQuery engine cannot find any result, it is adequate to provide an answer such as “Sorry, we cannot find the answer in the knowledge base.” In the current prototype, we provide only short answers to most questions, for example:

Ask QACPM> when will the task STF terminate?

……

QACPM Ans> 1/4/99

Alternatively, we can provide an answer in full sentences, such as “STF will terminate on 1/4/99.” This approach, however, increases the implementation complexity without providing additional information. Rather, using a short answer, such as “1/4/99,” is sufficient in most cases.

7. SYSTEM FRAMEWORK AND IMPLEMENTATION

A question answering system usually includes the following components: knowledge representation, question understanding, answer searching, and answer generation. Figure 7 illustrates the overall framework of our system. As shown in Figure 7, ifcXML is used to represent knowledge, while the XML query engine is employed for information query. In the first and most critical step, the system parses and understands the natural language questions. Natural language questions are then converted into XQuery expressions, which are executed by the XQuery engine. The query results are finally utilized by the generator to produce answers.

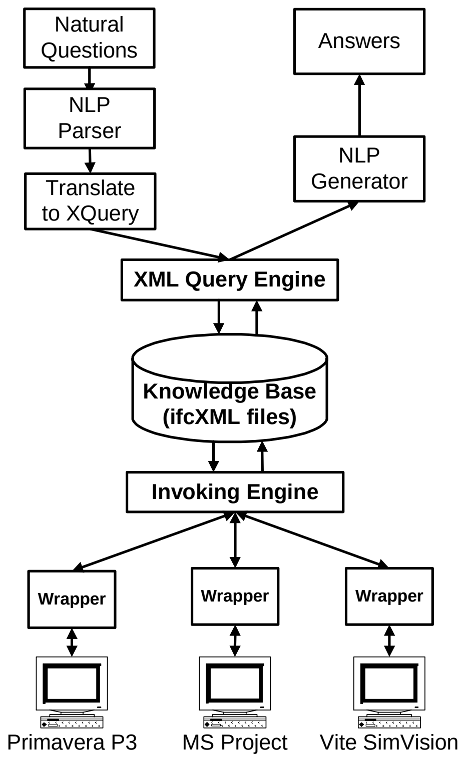


Figure 7: System Framework

Information in the knowledge base comes from various sources, such as Primavera Project Planner^{TM}(P3), Microsoft Project^{TM}, and Vite SimVision^{TM}. In addition to the wrappers to communicate with various software applications, the invoking engine is also needed to call the various wrappers.

The invoking mechanism is illustrated in Figure 8. Java socket communication is used as the protocol between ifcXML wrappers and the invoking engine. A communication agent, which includes an event listener, an event dispatcher, and a data mapper, is implemented for each ifcXML wrapper. The communication server is employed together with the invoking engine.

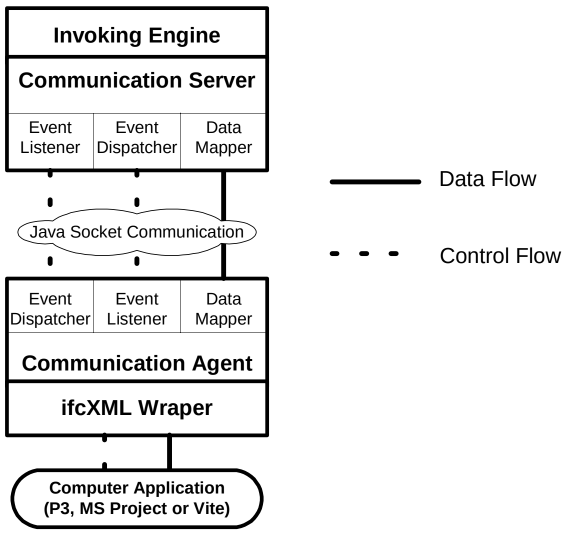


Figure 8: Invoking Framework

The messages in the system include control messages and data messages. Control messages are typically small in size, such as invoking and termination requests. However, data messages, such as the project scheduling information and organization information, are usually bigger in size. The event listener is responsible for receiving control messages, while the event dispatcher is used to send out control messages. The data mapper, in contrast, is responsible for sending and receiving data messages.

The detailed question answering process is illustrated in Figure 9. The rule generation process involves manual examination; however, this examination can be done by the experts in advance. At runtime, the system will first tag and parse the question using existing NLP tools. The result is the basis for further analysis. In particular, the object names in the ifcXML files can be used to help understand questions. In addition, the rules generated from XML trees and manual examination can also be used to help understand questions. Based on the analysis, we can match the question against potential rules. If no match is found, there is probably no answer for the question based on the existing knowledge. Otherwise, the system will do further analysis and generate answers.

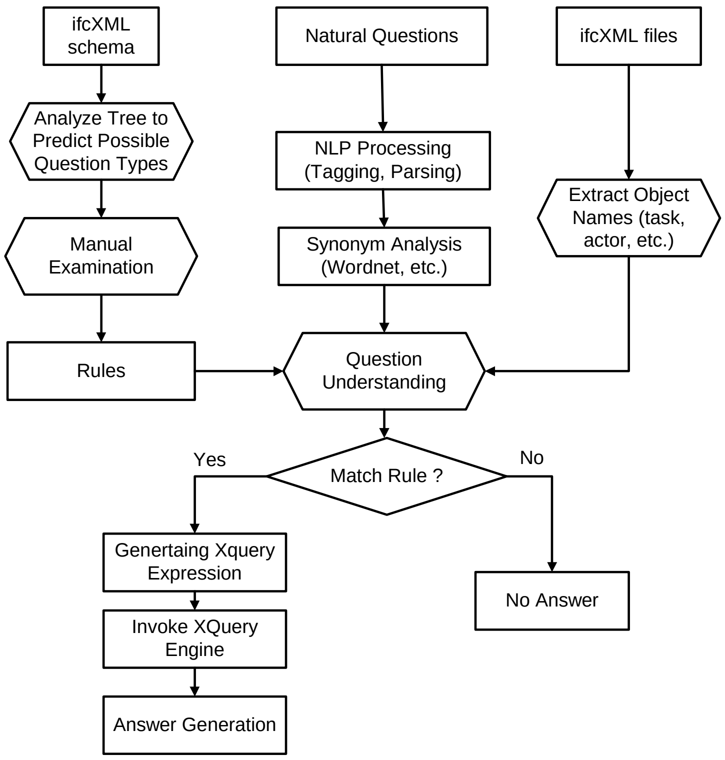

ifcXML schema
Natural Questions
ifcXML files
Analyze Tree to Predict Possible Question Types
NLP Processing (Tagging, Parsing)
Extract Object Names (task, actor, etc.)
Manual Examination
Synonym Analysis (Wordnet, etc.)
Rules
Question Understanding
Yes
Match Rule ?
No
Genertaing Xquery Expression
No Answer
Invoke XQuery Engine
Answer Generation

Figure 9: Detailed Process of the Question Answering System

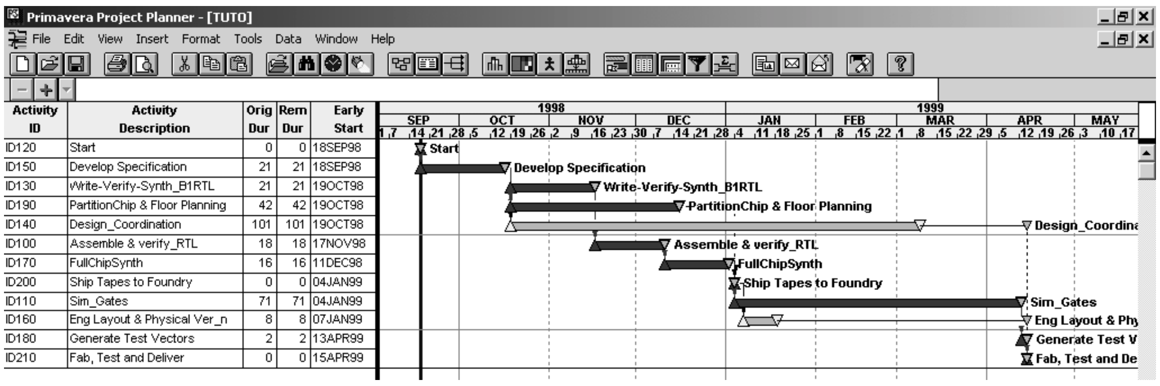
The prototype system is developed in Java. Various Java classes have been developed for different tasks, such as tagging questions and checking synonyms. The external programs, including XQuery engine and WordNet, are invoked via Java system calls, and results from these external programs are stored either in temporary files or Java InputStreams.

8. SAMPLE DEMONSTRATION

8.1 EXAMPLE PROJECT USED IN THE RESEARCH

In this project, we test our prototype system on a chip design project, which is a tutorial example in Vite SimVision™ (Vite 2000). The project involves both the design and the foundry staff to accelerate the design and construction of a new chip. The goal of the project is to design and fabricate a chip set for a new Personal Digital Assistant (PDA) product within a tight schedule. There are 12 activities in this project. Among the 12 activities are three milestone activities: "Start Project," "Ship Tapes to Foundry" and "Fab, Test and Deliver." The activity "Design_Coordination" maintains the overall control of the project.

Figure 10 shows the Gantt Chart of the Project in Primavera P3, where activities on the critical path are shown in dark color (red). Detailed scheduling information, such as the start dates, durations and finish dates of individual activities, is available in Primavera P3. In addition, dependency relationships among these activities are also included. A wrapper is developed to retrieve information from Primavera Project Planner™ (P3) and convert it into an ifcXML file (Cheng and Law 2002, Cheng et al. 2002).



Activity ID	Activity Description	Orig Dur	Rem Dur	Early Start
ID120	Start	0	0	18SEP98
ID150	Develop Specification	21	21	18SEP98
ID130	Write-Verify-Synth_B1RTL	21	21	19OCT98
ID190	PartitionChip & Floor Planning	42	42	19OCT98
ID140	Design_Coordination	101	101	19OCT98
ID100	Assemble & verify_RTL	18	18	17NOV98
ID170	FullChipSynth	16	16	11DEC98
ID200	Ship Tapes to Foundry	0	0	04JAN99
ID110	Sim_Gates	71	71	04JAN99
ID160	Eng Layout & Physical Ver_n	8	8	07JAN99
ID180	Generate Test Vectors	2	2	13APR99
ID210	Fab, Test and Deliver	0	0	15APR99

Figure 10. The Gant Chart of the Chip Design Project in Primavera P3

Figure 11 show the resulting ifcXML file from Primavera Project Planner™. In this example, the scheduling information is expressed using *WorkSchedule*, *Task* and *RelSequence* elements. In particular, a *Task* element in ifcXML maps to an activity in a project schedule. The *WorkSchedule* element is associated with the corresponding activity by using the same identifier as its identifier attribute. Similarly, the *RelSequence* element, which depicts the dependency relationships among activities, is associated with the predecessor and successor activities through its '*relatedProcess*' and '*relatingProcess*' attributes.

```
<WorkScheduleGroup>
    <WorkSchedule identifier="ID100" duration="18.0" freeFloat="0.0"
    totalFloat="0.0" startTime="11/17/1998" finishTime="12/10/1998"/>
</WorkScheduleGroup>
<TasksGroup>
    <Task taskid="ID100" description="Assemble and verify_RTL"/>
    <Task taskid="ID700" description="FullChipSynth"/>
</TasksGroup>
<RelSequenceGroup>
    <RelSequence id="depend0" relatingProcess="ID100"
    relatedProcess="ID170" timeLag="0.0"
    sequenceType="after-start"/>
</RelSequenceGroup>
```

Figure 11: Generated Sample ifcXML File from Primavera P3

Figure 12 shows the chip design project in Vite SimVision™. Actors and supervision relationships are shown in the top half of the display, while activities and dependency relationships are shown in the bottom half. Various aspects of the project, such as supervision, task assignment, communication and rework information, are represented using links in different colors in Vite SimVision™. Again, a wrapper is used to retrieve the organization information from Vite SimVision™ and convert it into the corresponding ifcXML file. To represent the information in Vite SimVision™, extensions are introduced in ifcXML. For example, we introduce the *Rework* element, which is not included in the current ifcXML schema, to represent the rework information among activities. The XML structure, *<Rework id="REW100" relatingTask="WVSB" relatedTask="PCAFP" />*, indicates that the failure of the task *WVSB* ("Write-Verify-Synth_B1RTL") will lead to the rework of the task *PCAFP* ("Partition Chip and Floor Planning").

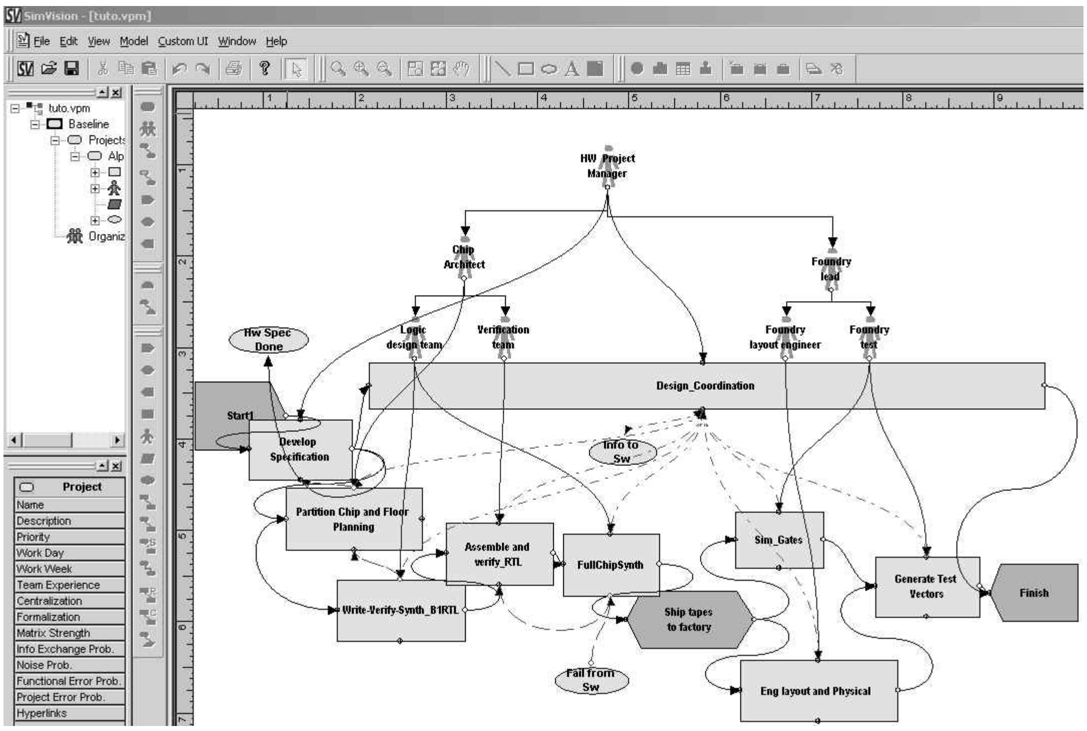


Figure 12: The Chip Design Project in Vite SimVision

8.2 Working Scenario

When invoked, the system first initializes and displays the welcome messages. First, users can choose to load any project in the database. The following command loads the TUTO project into the runtime environment:

Ask QACPM> load tuto

Currently, the project tuto has been loaded

There are 2 ifcxml files about the project :p3_tuto.xml, vite_tuto.xml

The ifcXML files will then be analyzed by the system, i.e., to extract object names. Once the program has finished the initialization process, it is ready to answer questions. The program first analyzes questions and converts them into XQuery expressions. The XQuery engine then executes the XQuery expressions based on the current knowledge base. The XQuery results are then parsed and presented to the user.

The system can answer various questions about project schedule, such as the start date, end date, duration, successors, and predecessors of an activity. The system will ignore the tense of the question. In addition, WordNet is used to identify synonyms. For example, in the following question, the system will give the same answer if we use the word “start” instead of “begin.”

```
Ask QACPM> when did Design_Coordination start?
Convert to XQuery expressions ...
for  $ws in document("p3_tuto.xml")//WorkSchedule
where    $ws/@identifier = "DC"
return   $ws/@startTime
XQuery Results:
 startTime="10/19/1998"
QACPM Ans> 10/19/1998
Ask QACPM> what is the duration of the task Generate Test Vectors?
Convert to XQuery expressions ...
for  $ws in document("p3_tuto.xml")//WorkSchedule
where    $ws/@identifier = "GTV"
return   $ws/@duration
XQuery Results:
 duration="2.0"
 QACPM Ans> 2.0
 Ask QACPM> which activity succeeds Sim_Gates?
 Convert to XQuery expressions ...
 for $ws in document("p3_tuto.xml")//RelSequence
 where $ws/@relatingProcess = "SG"
 return $ws/@relatedProcess
 XQuery Results:
  relatedProcess="GTV"
 QACPM Ans> Generate Test Vectors
```

It is also possible to ask other questions about the project, such as task assignment, supervision, cost, task uncertainty, and other information. Example usages are shown below:

```
Ask QACPM> who is responsible for Develop Specification?
Convert to XQuery expressions ...
for $ts in document("vite_tuto.xml")//RelAssignsTasks
where $ts/@relatingTask = "DS"
return $ts/@relatedActor
XQuery Results:
 relatedActor="HPM"
QACPM Ans> HW  Project Manager
Ask QACPM> what is the hourly cost of Foundry layout engineer?
Convert to XQuery expressions ...
for $ws in document("vite_tuto.xml")//Actor
where $ws/@id = "FLE"
return $ws/@salary
```

```
XQuery Results:
 salary="50.00"
QACPM Ans> 50.00
Ask QACPM> which task does sim_Gates need to communicate with?
Convert to XQuery expressions ...
for $cs in document("vite_tuto.xml")//Communication
where $cs/@relatedTask = "sim_Gates"
return $cs/@relatingTask
XQuery Results:
 relatingTask="DC"
QACPM Ans> Design_Coordination
```

In addition, the system can answer more complex questions by combining the information from Primavera Project Planner™ and Vite SimVision™. For example, to answer the first question *"which actors are involved with tasks on the critical path?"* the system searches from the Vite ifcXML file for task assignment information and searches from the Primavera P3 file for critical path information.

```
Ask QACPM> which actors are involved with tasks on the critical path?
Convert to XQuery expressions ...
for $ws in document("p3_tuto.xml")//WorkSchedule,
    $rs in document("vite_tuto.xml")//RelAssignsTasks
where $ws/@identifier = $rs/@relatingTask and $ws/@freeFloat="0.0"
return $rs/@relatedActor
XQuery Results:
 relatedActor="FT"  relatedActor="LDT"  relatedActor="HPM"  relatedActor="FLE"
 relatedActor="LDT" relatedActor="FT" relatedActor="CA"
QACPM Ans> Foundry test
Logic design team
HW  Project Manager
Foundry layout engineer
Logic design team
Foundry test
Chip Architect
Ask QACPM> which tasks on the critical path has the highest uncertainty?
Convert to XQuery expressions ...
for $ws in document("p3_tuto.xml")//WorkSchedule,
    $ts in document("vite_tuto.xml")//Task
where $ws/@identifier = $ts/@taskid and $ws/@freeFloat="0.0" and $ts/@uncertainty="high"
return $ws/@identifier
XQuery Results:
 identifier="DS"
QACPM Ans> Develop Specification
Ask QACPM> when does the activity with highest priority start?
Convert to XQuery expressions ...
for  $ws in document("p3_tuto.xml")//WorkSchedule,
     $ts in document("vite_tuto.xml")//Task
```

```
where    $ws/@identifier = $ts/@taskid and $ts/@priority = "high"
return   $ws/@startTime
XQuery Results:
startTime="9/18/1998"
QACPM Ans> 9/18/1998
```

8.3 ANALYSIS OF THE RESULT

There are two basic approaches to evaluating a question answering system: on-line and off-line evaluations (Breck et al. 1999). For on-line evaluations, system answers are judged by humans, while the answers in off-line evaluations are scored by an evaluation program against standard references.

In this research, the prototype system was tested on a selected set of questions in the project management domain. The system has been tested on the chip design project as well as a few residential building projects and demonstrates reasonable accuracy. However, the evaluation results depend on the selection of test questions as well as human judgement. The types, complexities, and ranges of questions, the human judgements on generated answers, the documents in the knowledge base, and other factors can all affect the performance of the system.

A standard evaluation program can save significant efforts in assessing a question answering system. However, since the results from project management tools are often stored in various internal formats, current evaluation programs cannot be directly applied in this research due to the characteristics of the project management domain. For example, Qaviar, an experimental evaluation program developed at the MITRE Corporation, judges the response using human generated answer keys and focuses on the Text REtrieval Conference (TREC) context (Breck et al. 2000). E-Rater, an evaluation system developed at ETS, is used to score essay questions of TOEFL takers (Burstein et al. 1998).

9. CONCLUSIONS

In this paper, we have presented a prototype question answering system in the project management domain. IfcXML is used as the knowledge representation format, while GNU Kawa Qexo is employed as the query engine. Once rules are generated from the ifcXML schema and examined by experts, the system requires no extra human involvement in building the knowledge base for individual project; thus, it can be easily adapted to different projects in the same domain. Information from the knowledge base, together with the syntactic and semantic analysis, is used for understanding natural questions. This approach can significantly help understand natural questions.

If the system can be extended to handheld devices, it would provide significant benefits for project management. For example, many construction projects are geographically distant from one another. As a result, many handheld devices, such as Palm Pilots and Pocket PCs, are often used by project members on construction sites. These handheld devices, however, cannot run most project management applications, such as Vite SimVison™, Primavera Project Planner™ and 4D Viewer, due to their small amount of memory and sub-optimal displays. A question answering system, on the other hand, requires little memory or displaying abilities on the client device. With a question answering system available, on-site personnel can simply input the question and get the information back using handheld devices.

Although this system has demonstrated the potential applications for NLP in project management, further research questions remain to be explored. The grammar used for the chart parser in this project is based on a heuristic approach and may not be complete. Thus, the chart parser may not be able to provide parse information for some types of questions, possibly resulting in incorrectly answers being generated. A more complete grammatical analysis of various questions may be necessary for a more robust system. In addition, further tests need to be performed to validate the prototype approach, for example, to include other application tools.

10. ACKNOWLEDGEMENTS

This work is partially sponsored by a Stanford Graduate Fellowship and the Product Engineering Program at NIST. The Product Engineering Program gets its support from the NIST's SIMA (Systems Integration for manufacturing Applications) program and the DARPA's Radeo Program. No approval or endorsement of any commercial product by the National Institute of Standards and Technology or by Stanford University is intended or implied. The second author would like to thank UK's Royal Academy of Engineering for the financial support provided to carry out this work.

REFERENCES

Abiteboul, S., Quass, D., McHugh, J., Widom, J., and Wiener, J. (1997), "The Lorel Query Language for Semistructured Data." *International Journal on Digital Libraries*, Vol. 1, No. 1, pp. 68-88.

Androutsopoulos, I., Ritchie, G. D., and Thanisch, P. (1995), "Natural Language Interfaces to Databases -- An Introduction." *Journal of Natural Language Engineering*, Vol. 1, No. 1, pp. 29-81.

Baeza-Yates, R., and Ribeiro-Neto, B. (1999), *Modern Information Retrieval*. Addison Wesley Ltd.

Brothner, P. (1998), "Kawa: Compiling Scheme to Java." *Lisp Users Conference*, Berkeley, CA. <http://sources.redhat.com/kawa/papers/KawaLisp98-html/index.html> (December 2002).

Burstein, J., Kukich, K., Wolff, S., Lu, C., and Chodorow, M. (1998). "Computer Analysis of Essays." *NCME Symposium on Automated Scoring*, April 1998.

Breck, E.J., Burger, J.D., House, D., Light, M., and Mani, I. (1999). “Question answering from large document collections.” *AAAI Fall Symposium on Question Answering Systems*, North Falmouth, MA.

Breck, E.J., Burger, J.D., Ferro, L., Hirschman, L., House, D., Light, M., and Mani, I. (2000). “How to Evaluate Your Question Answering System Every Day and Still Get Real Work Done.” *Proceedings of LREC-2000, Second International Conference on Language Resources and Evaluation*, Athens, Greece.

Callison-Burch, C., and Shilane, P. (2000), “A Natural Language Question and Answer System.” Unpublished Manuscript, Stanford University, Stanford, CA.

Cheng, J., and Law, K.H. (2002), “Using Process Specification Language for Project Information Exchange.” *3rd International Conference on Concurrent Engineering in Construction,* Berkeley, CA, pp. 63-74.

Cheng, J., Trivedi, P., and Law, K.H. (2002), “Ontology Mapping Between PSL and XML-Based Standards For Project Scheduling.” *3rd International Conference on Concurrent Engineering in Construction,* Berkeley, CA, pp. 143-156.

Diekema, A. R., Yilmazel, O., Chen, J., Harwell, S., He, L., Liddy, E. D. (2003) What Do You Mean? Finding Answers to Complex Questions. *Proceedings of the 2003 AAAI Spring Symposium: New Directions in Question Answering*. Palo Alto, California.

Diekema, A. Liu, X., Chen, J., Wang, H., McCracken, N., Yilmazel, O., and Liddy,E.D. (2000). Question Answering : CNLP at the TREC-9 Question Answering Track. *Proceedings of the 9th Text REtrieval Conference (TREC-9).* National Institute of Standards and Technology, Gaithersburg, MD.

Dym, C.L., and Levitt, R.E. (1991), *Knowledge-Based Systems In Engineering*. McGraw-Hill, Inc.

Fatdog. (2002), "XQEngine Introductory Tutorial." Fatdog Software, <http://www.fatdog.com/tutorial.html> (December 2002).

Fellbaum, C. (Editor), and Miller, G. (Preface) (1998), *WordNet: An Electronic Lexical Database (Language, Speech, and Communication)*. MIT Press.

Frank, G. (1999), “A General Interface for Interaction of Special-Purpose Reasoners within a Modular Reasoning System." *Question Answering Systems, Papers from the 1999 AAAI Fall Symposium*, pp. 57-62.

Froese, T., et al. (1999). "Industry Foundation Classes for Project Management-A Trial Implementation." *ITCON*, Vol. 4, pp. 17-36.

Genesereth, M.R., and Fikes, R. (1992), “Knowledge Interchange Format.” Version 3.0 Reference Manual, Computer Science Department, Stanford University, Stanford, CA.

Hovy, E., and Lin, C.Y. (1999), Automated Text Summarization in SUMMARIST. in: Mani, I. and Maybury M. (eds) *Advances in Automatic Text Summarization*. MIT Press.

IAI (1997), “Industry Foundation Classes.” Specification Volumes 1-4, International Alliance for Interoperability, Washington, DC.

IAI (2002), “Introduction to the IAI and it’s IFCs.” International Alliance for Interoperability. < http://radsite.lbl.gov/iai/IFC_2.0/iai/index.htm> (December 2002).

Jacobs, P., and Rau, L. (1990), “SCISOR: extracting information from on-line news.” *Communications of the ACM*, Vol. 33, No. 11, pp. 88-97.

Klein, D., and Manning, C. (2001a), “An O(n^3) Agenda-Based Chart Parser for Arbitrary Probabilistic Context-Free Grammars.” Technical Report, Computer Science Department, Stanford University, Stanford, CA.

Klein, D., and Manning, C. (2001b), “Parsing with Treebank Grammars : Empirical Bounds, Theoretical Models, and the Structure of the Penn Treebank.” *Proceedings of the 39th Annual Meeting of the ACL*, Toulouse, France, pp. 330-337.

Liebich, T. (2001), “XML schema language binding of EXPRESS for ifcXML.” MSG-01-001(Rev 4), International Alliance of Interoperability.

Marcus, M.P., Santorini B., and Marcinkiewica, M.A. (1993). "Building a large annotated corpus of English: the Penn Treebank." *Computational Linguistics*, Vol. 19, No.2, pp. 310-330.

Pereira, F. (1983), Logic for natural language analysis. Technical Note 275, SRI International.

Robie, J. (1999), “XQL Tutorial.” Software AG. <http://ibiblio.org/xql/xql-tutorial.html> (April 2002).

Schröder, I. (2002), “Ingo's Collection Of POS Taggers.” <http://nats-www.informatik.uni-hamburg.de/~ingo/icopost/> (May 2002).

Siméon, J. (2001), “Galax Implementation of XQuery.” XQuery Implementation Panel, XML 2001, Orlando. <http://db.bell-labs.com/galax/> (December 2002).

Sourceforge. (2002), “Sourceforge Xquench Project.” Open Source Development Network, <http://sourceforge.net/projects/xquench/> (May 2002).

Vasconcellos, M., and Leon, M. (1985), “SPANAM and ENGSPAN: Machine translation at the Pan American Health Organisation.” *Computation Linguistics*, Vol. 11, No. 2-3, pp. 122-136.

Vite. (2000), "SimVision Help." Vite SimVision Help Manual, Vite Corporation.

W3C (1998), “XML-QL: A Query Language for XML.” World Wide Web Consortium, <http://www.w3.org/TR/NOTE-xml-ql/ > (April 2002).

W3C (1999), “XML Path Language (XPath) Version 1.0.” World Wide Web Consortium, Recommendation 16.

W3C (2001), “XQuery 1.0: An XML Query Language,” W3C Working Draft 20.

Woods, W. A. (1973), Progress in Natural Language Understanding: An Application to Lunar Geology, *AFIPS Conference Proceedings*, Vol. 42, pp. 441-450.

Young, M.J. (2001), *Step by Step XML*. Microsoft Press.

Zajac, R. (2001), "Towards Ontological Question Answering." *ACL Open Domain Question Answering Workshop*, Toulouse.