

# A QUESTION ANSWERING SYSTEM FOR CONSTRUCTION PROJECT MANAGEMENT

Jinxing Cheng<sup>1</sup>

## ABSTRACT

The usage of computer applications in the construction industry has steadily increased over the years, as has the complexity of many applications. However, project managers, who are usually responsible for making decisions, are not necessarily familiar with these ever-increasingly complex applications. As a result, a question answering system is needed for efficiently managing construction projects. In this paper, we examine the various aspects involved in building a question answering system. In particular, we use ifcXML files as the knowledge representation system, which require no manual efforts to build the knowledge base. Moreover, we explore the mechanism to utilize information in the knowledge base for question understanding. A prototype question answering system, based on various technologies, has been built and successfully tested on several construction projects.

## KEYWORDS

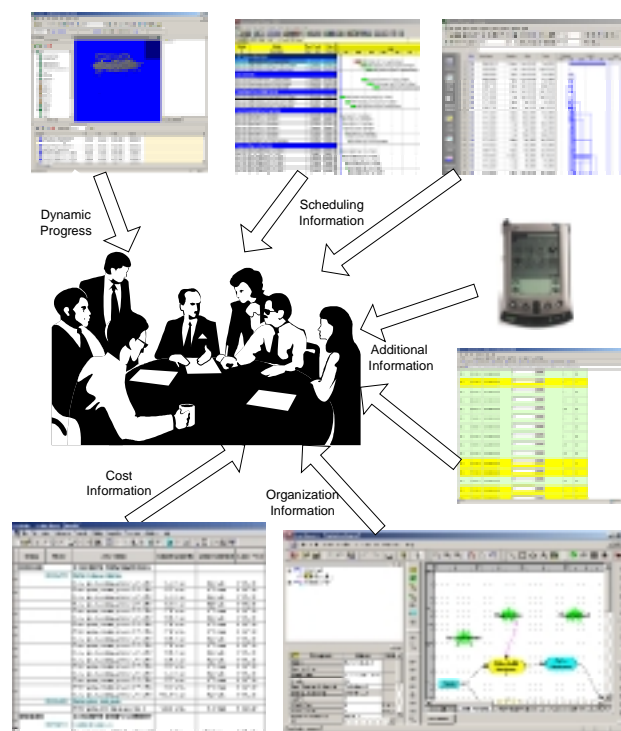
Question Answering, XQuery, ifcXML

---

<sup>1</sup> PhD Student, Civil and Environmental Engineering Department, Stanford University, Stanford, CA 94305-4020, email: [cjx@stanford.edu](mailto:cjx@stanford.edu)

## 1. MOTIVATION

During the past two decades, many applications are being employed for managing increasingly complex construction projects. For example, as shown in Figure 1, different members of a project team may use Primavera Project Planner (P3)<sup>TM</sup> or Microsoft Project<sup>TM</sup> to schedule the project, Vite<sup>TM</sup> to simulate the project organization, Timberline's Precision Estimating<sup>TM</sup> to estimate project cost, and 4D Viewer (McKinney and Fischer 1998) to view the progress of construction. On the other hand, due to the advancements in information technologies, these applications are getting more and more complex. Project team members, however, are not necessarily familiar with these new technologies and applications. Therefore, a question answering system is necessary, with which project team members can easily query project information in a natural interface.



**Figure 1: Various Applications in Construction Project Management**

Currently, question answering systems are available for certain predefined domains. Callison-Burch and Shilane developed a question answering system in the family tree domain (Callison-Burch and Shilane 2000). The system used Knowledge Interchange Format (KIF) (Genesereth and Fikes 1992) files as the knowledge representation system, and used Java Theorem Prover (JTP) to infer answers. However, the knowledge base was created manually; thus, it is hard to generalize the system to other domains. Moreover, even within the same domain, the system is not scalable, since different projects contain different information.

Zajac used an ontology-based semantic approach for question answering (Zajac 2001). Both the questions and source texts are parsed into partial underspecified semantic expressions. This approach, however, seems to assume that the source texts are also in natural sentences, which usually is not true. In many construction applications, project

information is usually either in plain text files or in semi-structured data files, none of which are natural sentences.

Efforts have also been made in developing natural language interfaces to databases. Androutsopoulos et al. discussed various methods and solutions available in translating natural questions to database queries (Androutsopoulos et al. 1995). Although some solutions seem promising in a small domain, it is difficult to apply them to the construction industry, where database interfaces are not supported in many applications.

In summary, a question answering system is needed for construction project management. This study uses ifcXML files as the knowledge representation system, which requires no manual work. In addition, this study explores the ways to utilize the knowledge base to predict and understand questions. Finally, a prototype question answering system in the project scheduling domain has been built and successfully tested on a few practical construction projects.

## **2. INTRODUCTION TO RELATED TECHNOLOGIES**

In this project, various technologies have been used, such as XQuery, GNU Kawa's Qexo, POS tagging tool, chart parsing tool, and WordNet. In the following sections, we will give a brief introduction about these 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, however, IFC has been extended to support data exchange for cost estimating and project management purposes (Froese et al. 1999). In brief, based on EXPRESS, IFC is designed to exchange data among Architecture, Engineering, Construction and Facilities Management (AEC/FM) applications.

XML (Extended 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. In particular, ifcXML is one of the most popular XML schemas in the A/E/C domain.

Currently, IfcXML is automatically translated from an IFC/EXPRESS source (Liebich 2001). A fairly extensive schema with over 400 pages, IfcXML enables the exchange of IFC data alternatively in XML format. IfcXML not only deals with geometry and product data, but also supports life cycle project information including architecture, HVAC, construction and facility management.

### **2.2 XML QUERY LANGUAGE**

XML query languages are used to query information from XML files. We have investigated many XML query languages, such as XQL (Robie 1999), XML-QL (W3C 1998), LOREL (Abiteboul et al. 1997), Xpath (W3C 1999) and XQuery (W3C 2001). Our investigation shows that XQuery is the most appropriate language for the research purpose.

Xquery (W3C 2001) is an XML query language jointly defined by the XML Query Working Group and the XSL Working Group. XQuery is designed to be applicable for all types of XML data sources.

XQuery uses a syntax similar to SQL. For example, the following XQuery sentences are used to query the start date of the activity ELPV:

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

### 2.3 XML QUERY ENGINE

Although XQuery is still in W3C Working Draft version, many vendors have implemented XQuery, such as Microsoft XML Query (Microsoft 2002), SourceForge Xquench (SourceForge 2002), FatDog XML Query Engine (FatDog 2002), Lucent Galax (Lucent 2002), X-Hive Query(X-Hive 2002), and GNU Kawa Qexo (Kawa 2002). After investigating these XQuery implementations, the GNU Kawa Qexo was used as the query engine in the project.

Qexo, an open source project, is a partial implementation of the XML Query language from GNU Kawa (Kawa 2002). There are two methods to use Qexo. Qexo can be used in an interactive environment, where users can input query sentences at the command line. In addition, Qexo can compile XQuery sentences into Java byte codes, which will significantly improve the query efficiency.

### 2.4 POS TAGGING AND CHART PARSING TOOLS

There are many POS tagging tools available. In this project, the trigram tagger from ICOPOST was used. ICOPOST (Schröder 2002) is a set of free POS taggers, which include Maximum Entropy Tagger (MET), Trigram Tagger (T3), and Error-driven Transformation-based Tagger (TBT). T3 is based on Hidden Markov Models (HMM).

The context-free grammar chart parser developed at Stanford University was used in the project. 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 outputted as the parsing results.

### 2.5 WORDNET AND JWORDNET

Developed by the Cognitive Science Laboratory at Princeton University, WordNet (WordNet 2002) 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, such as Microsoft Windows and Unix.

JWordNet (Steele 2002) is a Java interface to lexical relationships in WordNet. Using JWordNet, Java programmers can create portable Java applications. JWordNet provides a few classes, such as DictionaryDatabase, IndexWord and Synset, which can be used to navigate the lexical information of any word in WordNet.

In this Project, JWordNet was first used. However, since JWordNet seems to be unstable and a little buggy sometimes, WordNet was ultimately employed.

## 3. KNOWLEDGE REPRESENTATION

Knowledge representation is crucial in building a question answering system. Ideally, the knowledge base should be automatically built based on existing information in the domain. In the project scheduling domain, project information is usually stored in various applications in many internal formats; however, wrappers are available, which can

retrieve project information from various sources and convert the information into standard formats, such as ifcXML.

In this research, we use ifcXML as the knowledge representation system. Project information in various applications can be extracted into ifcXML files using wrappers.

In this research, we focus on the project scheduling domain; thus, only a small portion of the ifcXML schema is used. In particular, 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. Figure 2 shows a snapshot of a sample ifcXML file.

```
<WorkScheduleGroup>
  <WorkSchedule identifier="ID100" duration="18.0" freeFloat="0.0"
totalFloat="0.0"          startTime="12/17/1998" finishTime="1/4/99"/>
</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 2: Sample ifcXML file

## 4. PARSING AND UNDERSTANDING NATURAL QUESTIONS

### 4.1 ANALYZING IFCXML TREES

Understanding natural questions is rather difficult for computer applications, which is partly due to the fact that there are too many varieties of natural questions. Even if we limit our research into a small predefined domain, for example, the project scheduling domain, there are still significant varieties of possible questions. A few sample questions are listed as follows:

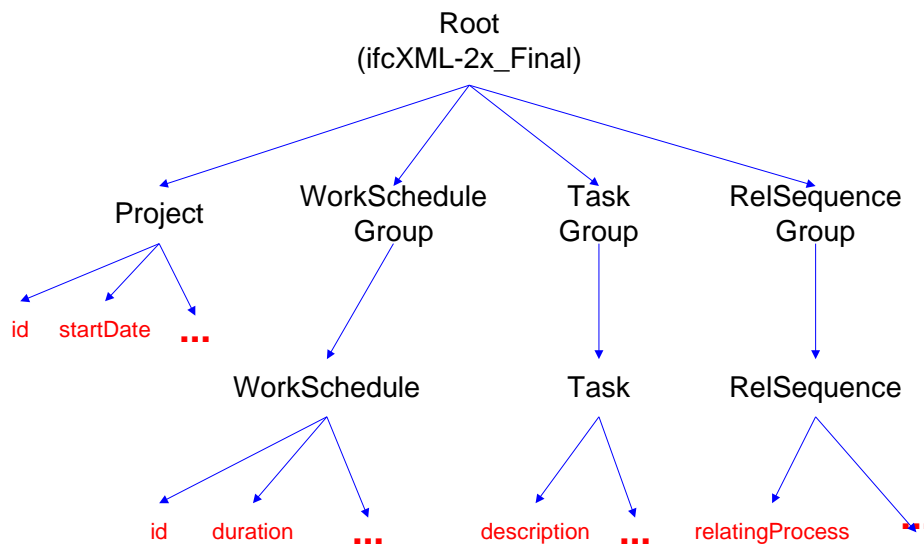
1. *When did the activity PouringConcrete starts?*
2. *Why should activity PouringConcrete start before ErectingBeams?*
3. *Which subcontractor submitted scheduling changes yesterday?*

Although all these questions are (1) syntactically correct; (2) semantically sound; and (3) within the project scheduling domain, not all these questions can be answered. In particular, we do not have information for questions 2 and 3 in our ifcXML files. Nonetheless to say, there are many questions, which can be either syntactically incorrect or semantically unsound.

On the other hand, ifcXML schema can be utilized for question understanding. Using the information in the schema, we can predict what kind of questions we can answer. For the questions with no answer in the knowledge base, we can safely ignore them; in other words, we do not need to understand the exact meanings of many questions we cannot answer.

The first step is to analyze the tree structure of ifcXML files. We have developed a Java program, which can analyze all 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 into rules. Each rule contains one relation word and several parameters, such as the rules (duration activity) and (finish activity). Usually, the first word in the rule specifies the relation, while the remaining words represent the parameters in the rule. Figure 3 shows the tree structure of the ifcXML files. The leaf nodes are attributes, and all other nodes are elements. In addition, not all attributes are shown in Figure 3.



**Figure 3: Tree Structure of IfcXML Files**

Obviously, there are several types of questions that the system should be able to answer, such as:

- Questions asking the attribute value of an element.
- Questions asking which element has certain attribute value.
- Some questions which involve several elements.

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

The third type of questions, on the other hand, is quite complex. We first need to figure out the relationship among different elements; from that, we need to predict the possible questions we can answer, and generate the corresponding rules for the questions. In addition, some generated rules may not make sense. Again, these rules need to be examined manually. One sample rule is (description (hasDuration number)). For this rule, the possible question is to ask the description of the activity that has certain 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 worthy the efforts. Since the ifcXML files for different projects have similar XML tree structures, we only need to select one typical ifcXML file and manually examine the rules once. The rules can then be used for different 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 know whether a word is a noun or verb. The POS information can be used to help understanding the meaning of words. In particular, it can help us to identify whether or not a word is a potential relation or parameter in a rule.

Questions need to be processed before we can tag it. In particular, we need to separate the word 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 ? .*

We can notice that the POS tagger does not always provide correct POS information. For example, it tags the word “start” in the first sentence as a noun, which actually is a verb. Nonetheless, the POS information provides a good basis for understanding questions.

## 4.3 PARSING QUESTIONS

In this project, a context-free grammar chart parser is used for parsing questions. To use the chart parser, we first need to create a grammar and lexicon files. While it is possible to write a grammar file for questions in the project scheduling domain, it is tedious if not impossible to create a lexicon file for all possible questions, since the lexicon file needs to contain all words which can appear in the questions.

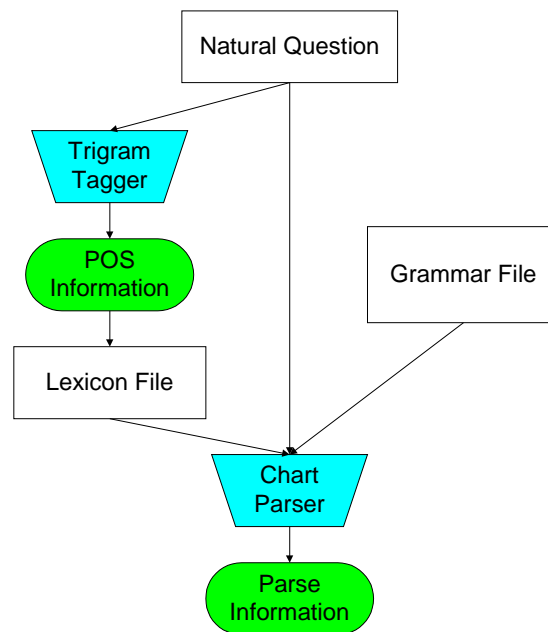
We can extract a grammar file from Penn Treebank. However, the extracted grammar file is huge, while most of the grammar rules are not useful for question sentences. Thus, the volume of grammar rules will significantly decrease the parsing speed. On the other hand, a much smaller grammar file is possible for questions in a small domain. As a result, a grammar file for questions in the project scheduling domain is developed. Figure 4 shows part of the grammar file, which is largely based on a heuristic approach.

```
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
VP -> VBG %%0.1
VP -> VBN %%0.1
VP -> VBZ %%0.1
? -> ? %%0.1
.....
```

Figure 4: Grammar File for the system

For the lexicon file, we can also extract lexicons from Penn Treebank. The standalone program ExtractPTBRules (Stanford 2002) developed by Stanford Natural Language Process group can be used to extract lexicon files from a collection of Penn Treebank sentences. However, even if we extract lexicons from a large collection of document, many words in the questions can still be not included in the lexicon file; as a result, the chart parser cannot find any parse for the question.

Another simple but effective approach is to utilize the POS tagging results. We can generate a lexicon file based on the tagging results. Obviously, all words in the question will appear in the lexicon file. Assuming the question is syntactically correct, the chart parser can always find a parse for the question. Figure 5 illustrates this approach.



**Figure 5: 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. Table 1 illustrates the process of generating a lexicon file for the question, while Figure 6 shows the parsing results.

**Table 1: Prepare Lexicon File**

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

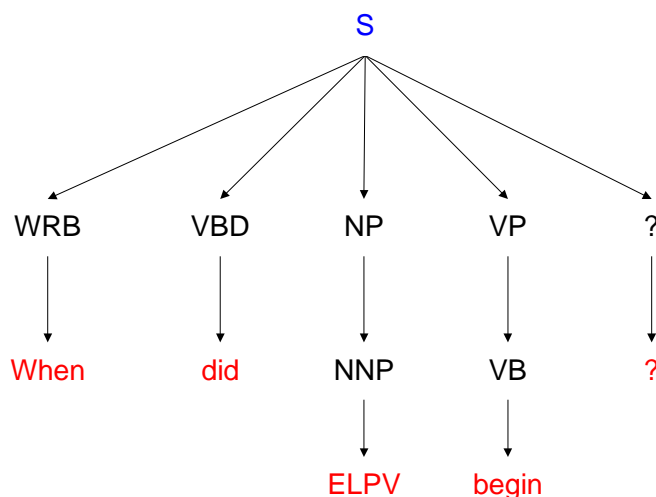


Figure 6: Parse Tree of A Sample Question

#### 4.4 ANALYZING CONCEPTS AND CATEGORIZING QUESTIONS

In this project, ifcXML files are used as the knowledge representation system; as a result, useful information can be obtained from the ifcXML files before we actually start processing questions. For example, we can parse the ifcXML files and store all activity names into a list. When the system encounters a question later, it can search in the list; if a word is found in the activity list, it is a strong indication that this word represents an activity name. Thus, it will significantly help the system to understand the questions.

In the previous sections, we have discussed possible questions we can answer from the knowledge base, and how to express them in rules. In addition, we have discussed how to obtain the POS and parsing information of the questions. Based on the information above, we can analyze the concepts in the questions and categorize the questions into different rules. Usually, the following words are the most important for understanding questions: question word, nouns and verbs.

- Question words, such as when, what, how and does, determine the type of the question; in addition, it also implies what kind of answer the user expects.
- 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 questions:

*What is the start date of ELPV?*

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 example, in the following sentence:

*When does ELPV end?*

The verb end is a strong indication that this sentence should be categorized into rule (end activity), where ELPV corresponds to parameter activity 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, however, we can categorize both situations into the rule (*start activity*).

## 5. SEARCHING ANSWERS IN THE KNOWLEDGE BASE

Translating questions into XQuery expressions is not an easy task. Therefore, a Java program is 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 $ws in document("$xmlfile")//WorkSchedule
where $ws/@identifier = "$I"
return $ws/@duration
```

There are two parameters in the XQuery expressions. The first parameter \$xmlfile appears in all rules; it represents the current knowledge base (the ifcXML file) we are using. The second parameter \$I corresponds to the activity in the rule.

When a question is parsed, it will be categorized into a rule based on the syntactic and semantic information of the question. For example, the question “What is the duration of ELPV ?” will be categorized into the rule (*duration activity*), where the value of the parameter *activity* is ELPV. Suppose that the current knowledge base is tuto.xml, the following XQuery expressions will be generated for this question:

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

The XQuery will be compiled into Java byte codes by the Qexo query engine. We can then use the generated codes to search the answer in the knowledge base.

## 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 example, for a *wh-question*, a question with specific information is expected. For a *how 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 enough.

In most cases, for a *wh-question*, if the XQuery engine cannot find any result, it is enough to provide an answer such as “Sorry, we can not find the answer in the knowledge base.”

In this research, we prefer to give short answers for most questions, for example:

```
Ask QACPM> when will the task STF terminate?
```

```
.....
```

```
QACPM Ans> 1/4/99
```

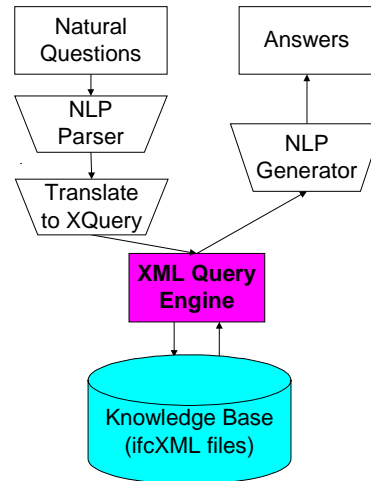
We can provide an answer in full sentence, such as “STF will terminate on 1/4/99.” This approach, however, is error prone. Rather, using short answer, such as “1/4/99”, is sufficient in most cases.

## 7. SYSTEM FRAMEWORK AND IMPLEMENTATION

### 7.1 SYSTEM FRAMEWORK

A question answering system usually includes the following components: question understanding, knowledge representation, answer searching, and answer generation. Figure 3 exhibits the framework of our system. As shown in figure 7, ifcXML files are used as the knowledge representation system, while the XML query engine is employed

for information query. The system first parses and understands human questions, which is one of the most critical steps. Human questions will then be converted into XQuery expressions, which will be executed by the XQuery engine. The query results will be utilized by the generator to produce natural answers.



**Figure 7: System Framework**

## 7.2 SYSTEM IMPLEMENTATION

The prototype system is developed in Java. Various Java classes have been developed for different tasks, such as tagging questions and checking synonyms. The system can dynamically analyze the information in the knowledge base, such as activity names; the information can then be used by other classes for understanding and categorizing questions. For example, the analyzation results are useful for generating XQuery expressions. The following code (Figure 8) shows the process of generating XQuery expressions:

```

String line;
while( (line = rd.readLine())!= null ){

    if(line.equalsIgnoreCase(rule)){
        line = rd.readLine();
        while( (line = rd.readLine())!=null) {

            if( line.startsWith("*****") ) break;
            line = Util.replace(line, "$xmlfile", xmlfile);

            for(int i=0; i<params.size();i++){
                int j = i+1;
                String arg = "$"+ j;
                String param = (String)params.elementAt(i);
                line = Util.replace(line, arg, param);
            }
            out.println(line);
        }
    }
}

```

**Figure 8: Sample Code of Generating XQuery Expressions**

In the prototype system, various external programs have been used, such as XQuery engine, WordNet and others. The external programs are invoked via system calls, and

results from these programs are stored either in temporary files or `InputStreams`. For example, as shown in Figure 9, the following code shows the procedure of invoking XQuery Engine:

```
String cmd1 = "java -jar kawa-1.6.98.jar --xquery --main -C " +
xqlFile;
String xqlClass = xqlFile.substring(0, xqlFile.lastIndexOf('.'));
String jmd = "java -cp .:kawa-1.6.98.jar " + xqlClass + " >
tmpxql.rs";
String cmd2[] = { "/bin/sh",
"-c",
jmd};
Runtime.getRuntime().exec(cmd1).waitFor();
Runtime.getRuntime().exec(cmd2).waitFor();
```

**Figure 9: Sample Code of Invoking XQuery Expression**

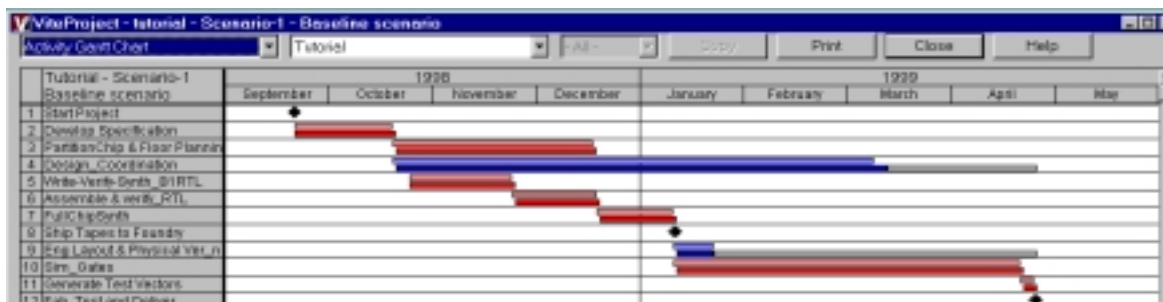
## 8. SAMPLE DEMONSTRATION

### 8.1 CONSTRUCTION PROJECTS TESTED IN THE RESEARCH

In this project, we test on two sample projects:

- Chip Design project (tuto.xml)
- Mortenson Ceiling Project (ceil.xml)

The chip design project (Figure 10) is selected from Vite; 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 there are three milestone activities: “Start Project,” “Ship Tapes to Foundry” and “Fab, Test and Deliver.” The activity “Design\_Coordination” is to maintain the overall control of the project. Wrappers are used to retrieve the scheduling information from Vite and convert it into an ifcXML file.



**Figure 10: The Gantt Chart of the Chip Design Project in Vite**

Mortenson Ceiling Project is part of the Walt Disney Concert Hall, built by Mortenson Construction, and designed by Frank O.Gehry & Associates. There are 191 activities and 459 dependency relationships in the project. Figure 11 shows the project schedule in Primavera Project Planner.

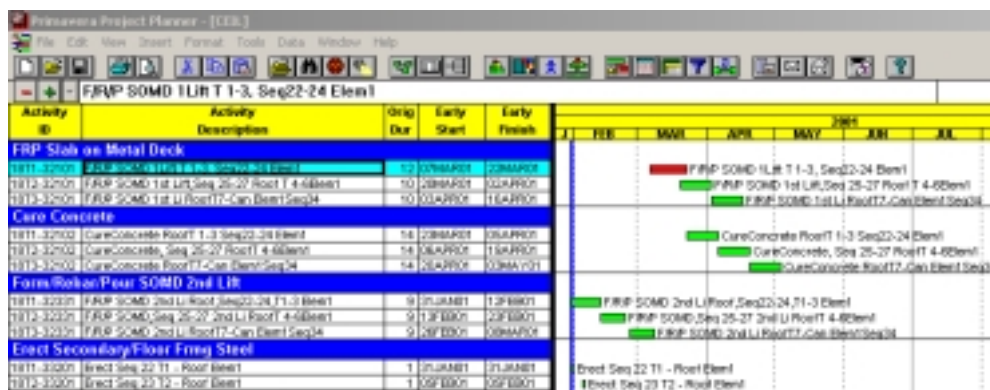


Figure 11: Project Schedule in Primavera Project Planner

## 8.2 WORKING SCENARIO

The system first starts initialization and displaying welcome messages when invoked. Once the program finished initialization, it can start accepting questions. The program first analyzed questions and converted them into XQuery expressions. The XQuery engine then executed the XQuery expressions on the current knowledge base. The XQuery results were parsed and presented to the user.

*Ask QACPM> When did the activity ELPV start?*

*Convert to XQuery expressions ...*

*for \$ws in document("tuto.xml")//WorkSchedule*

*where \$ws/@identifier = "ELPV"*

*return \$ws/@startTime*

*XQuery Results:*

*startTime="4/15/1999"*

*QACPM Ans> 4/15/1999*

The system will ignore the tense of the question. In addition, WordNet was used to identify synonyms. For example, in the following question, the word “begin” was used instead of “start”; however, we got the same answer.

*ask QACPM> when does ELPV begin?*

*Convert to XQuery expressions ...*

*for \$ws in document("tuto.xml")//WorkSchedule*

*where \$ws/@identifier = "ELPV"*

*return \$ws/@startTime*

*XQuery Results:*

*startTime="4/15/1999"*

*QACPM Ans> 4/15/1999*

We can ask other information about the project, such as the duration, end date, and description of the activities. The example usages are shown as follows:

*Ask QACPM> what is the duration of ELPV?*

*Convert to XQuery expressions ...*

*for \$ws in document("tuto.xml")//WorkSchedule*

*where \$ws/@identifier = "ELPV"*

*return \$ws/@duration*

*XQuery Results:*

*duration="8.0"*

*QACPM Ans> 8.0*

*Ask QACPM> What is the description of the task ELPV?*

*Convert to XQuery expressions ...*

*for \$ts in document("tuto.xml")//Task*

```

where $ts/@taskid = "ELPV"
return $ts/@description
XQuery Results:
description="Eng Layout and Physical Ver_n"
QACPM Ans> Eng Layout and Physical Ver_n

```

During the process, we can change to other construction projects. For example, we can use the command “select ceil.xml” to change to the Mortenson Ceiling Project.

```

Ask QACPM> select ceil.xml
Currently ceil.xml has been loaded

```

The command “list acts” can be used to retrieve all activities in the current construction project, for example:

```

Ask QACPM> list acts
00H0-8011F
00H0-8011S
00H0-8012D
1110-80101
1120-80101
1130-80101
1610-33221
.....

```

We can also ask other kind of questions, such as *does* and *how many* questions.

```

Ask QACPM> Is the activity 1610-33221 the predecessor of 1620-33221?
Convert to XQuery expressions ...
for $ws in document("tuto.xml")//RelSequence
where $ws/@relatingProcess = "$2" and $ws/@relatedProcess = "$1"
return $ws/@id
XQuery Results:
id="depend5"
QACPM Ans> Yes.

```

```

Ask QACPM> How many successors does the activity 1610-33221 have?
Convert to XQuery expressions ...
for $ws in document("ceil.xml")//RelSequence
where $ws/@relatingProcess = "1610-33221"
return $ws/@relatedProcess
XQuery Results:
relatedProcess="1610-83251" relatedProcess="1620-33221"
QACPM Ans> 2

```

## 9. LIMITATIONS

### 9.1 COVERAGE OF QUESTIONS

Basically, the coverage of questions in the prototype system is limited by the information we have in the knowledge base. In particular, it is limited by the scheduling information in the ifcXML files. Currently, our system covers three basic types: *Wh-question*, *how-questions*, and *is/dose* questions.

- Wh-Questions:  
*When did ID100 finish?*  
*What is the duration of the activity ELPV?*  
*When did the task ELPV begin?*
- How Questions:

*How many successors does the activity ID100 have?*

- Yes or No Questions:  
*Is the activity ID100 the predecessor of ID120?*

However, there are many complex questions, which are not covered in the prototype system. Even within the above categories, some questions can not be handled by the prototype system, for example:

*What is the start date and duration of ID100?*

The knowledge base has the information to answer the above questions. The current prototype system, however, lacks the ability to divide the question into two rules: (*duration activity*) and (*start activity*).

## **9.2 DATE COMPARISON AND CALCULATION**

Date information is very important in construction projects. XQuery, however, does not directly support this data type; thus we cannot use XQuery to compare dates directly. Rather, additional work needs to be involved in comparing the dates. Currently, the prototype system does not support date comparison and calculation; for example, it cannot handle the following questions:

*Which activity starts earlier, ID100 or ID120?*

*Which activity starts on December 25, 1998?*

## **9.3 OTHER LIMITATIONS**

Currently, the grammar used for the chart parser is based on the observation from collected sample questions. Obvious, it may not be complete. Further research need to be done on the grammatical analysis of various questions.

In addition, the rule generation and the translation to XQuery expressions cover only part of the possible questions. Moreover, the current answer generator is simply based on heuristic observations. To make the answers more natural, further research need to be conducted.

## **10. CONCLUSIONS**

In this paper, we have successfully developed a question answering system in the project scheduling domain. In particular, ifcXML files are used as the knowledge representation system, while GNU Kawa Qexo is employed as the query engine. The system requires no human involvement in building the knowledge base; thus it can be easily adapted to different construction projects. Information from the knowledge base, together with the syntactic and semantic analysis, is used for understanding natural questions. This approach can significantly help understanding natural questions. Furthermore, the system has been successfully tested on several construction projects.

Although this system has demonstrated its potential for construction project management, further research needs to be done. In particular, the grammar used for the chart parser in this project is based on a heuristic approach; thus, a more complete grammatical analysis of various questions is necessary for a robust system.

## 11. ACKNOWLEDGEMENTS

The author would like to thank Professor Christopher Manning and Andrea Tompa, Stanford University, for their valuable comments and suggestions about the research. The model of the Mortenson Ceiling Project is provided by Professor Martin Fischer and his research group at Stanford University.

## 12. 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*, 1(1):68-88, April 1997
- Androustopoulos I., Ritchie G. D., and Thanisch P., "Natural Language Interfaces to Databases -- An Introduction." *Journal of Natural Language Engineering*, vol.1, no.1, Cambridge University Press, 1995.
- Callison-Burch C. and Shilane P. (2000), "A Natural Language Question and Answer System." CS224N course final project, June 3, 2000
- FatDog (2002), FatDog XML Query Engine, <http://www.fatdog.com/> (Accessed: 20 May 2002), Fatdog Software
- Froese, T., Fischer, M., Grobler, F., Ritzenthaler, J., Yu, K., Sutherland, S., Staub, S., Akinci, B., Akbas, R., Koo, B., Barron, A., and Kunz, J., (1999). "Industry Foundation Classes for Project Management-A Trial Implementation." *ITCON*, Vol. 4, 17-36.
- Genesereth M.R. and Fikes R. (1992), "Knowledge Interchange Format Version 3.0 Reference Manual." Computer Science Department, Stanford University.
- IAI (1997), "Industry Foundation Classes." Specification Volumes 1-4, International Alliance for Interoperability, Washington DC 1997.
- Kawa(2002), Kawa-XQuery, <http://www.gnu.org/software/qexo/> (Accessed: 20 May 2002), Per Bothner
- Liebich T. (2001), "XML schema language binding of EXPRESS for ifcXML," MSG-01-001(Rev 4), International Alliance of Interoperability, 2001.
- Lucent (2002), Lucent Galax, <http://db.bell-labs.com/galax/> (Accessed: 20 May 2002), Lucent Technologies
- McKinney, K. and Fischer, M. (1998) "Generating, evaluating and visualizing construction schedules with 4D-CAD tools." *Automation in Construction*, 7(6), 433-447.
- Microsoft (2002), "Microsoft XML Query Language Demo." <http://131.107.228.20/> (Accessed: 20 May 2002), Microsoft Corporation.
- Robie J. (1999), "XQL Tutorial." <http://ibiblio.org/xql/xql-tutorial.html> (Accessed: 25 April 2002), Software AG.
- Schröder I.(2002), "Ingo's Collection Of POS Taggers." <http://nats-www.informatik.uni-hamburg.de/~ingo/icopost/> (Accessed: 25 May 2002)
- Sourceforge (2002), "Sourceforge Xquench project." <http://sourceforge.net/projects/xquench/> (Accessed: 20 May 2002), Open Source Development Network.
- Stanford University (2002), Extract PTB Rules, <http://www-nlp.stanford.edu/nlp/javadoc/javanlp/edu/stanford/nlp/lexgram/ExtractPTBRules.html> (Accessed: 28 May 2002), Stanford Natural Language Processing Group, Stanford University.
- Steele O. (2002), "JwordNet." <http://jwn.sourceforge.net/> (Accessed: 25 May 2002)

- W3C(1998), "XML-QL: A Query Language for XML." World Wide Web Consortium, 1998, <http://www.w3.org/TR/NOTE-xml-ql/>, (Accessed: 25 April 2002).
- W3C(1999), "XML Path Language (XPath) Version 1.0." World Wide Web Consortium, Recommendation 16 November 1999.
- W3C(2001), "XQuery 1.0: An XML Query Language," W3C Working Draft 20, December 2001
- WordNet, <http://www.cogsci.princeton.edu/~wn/> (Accessed: 25 May 2002), Princeton University
- X-Hive (2002), X-Hive's XQuery, <http://217.77.130.189:8080/demos/xquery/index.html> (Accessed: 20 May 2002), X-Hive Corporation
- Young, M.J. (2001), *Step by Step XML*, Microsoft Press, 2001.
- Zajac, R. (2001), "Towards Ontological Question Answering.", *ACL Open Domain Question Answering Workshop*, July 6, 2001, Toulouse.