

A Knowledge-Based Approach to Question-Answering

Peter Clark, John Thompson	Bruce Porter
Knowledge Systems	Dept Computer Science
Maths and Computing Technology	Univ. of Texas at Austin
Boeing, PO Box 3707, Seattle, WA	Austin, TX 78712
{peter.e.clark,john.a.thompson}@boeing.com	porter@cs.utexas.edu

Abstract

Our long-term research goal is to create systems capable of answering a wide variety of questions, including questions which were unanticipated at the time the system was constructed, and questions tailored to novel scenarios which the user is interested in. Our approach is to augment on-line text with a knowledge-based question-answering component, capable of reasoning about a scenario which the user provides, and synthesizing customized answers at run-time to his/her questions. To answer a question, the system creates a model of the user's scenario of interest, infers the facts about it required for the answer, assembles those facts into a single structure, and presents it to the user. This process is guided by "answer schemata" (one for each different question type), specifying what information should be included in the answer and how it should be organized. In this paper, we describe an implemented system based on this approach, which has been applied to several different application domains. We conclude with a discussion of the strengths and limitations of this approach.

Introduction

Our long-term research goal is to create systems capable of answering a wide variety of questions, including questions which were unanticipated at the time the system was constructed, and questions tailored to novel scenarios which the user is interested in. Our approach is to augment existing on-line text resources (e.g. manuals, documentation), which are capable of answering the more routine questions already, with a inference-capable question-answering component, allowing the system to infer answers to users' questions which are outside the scope of the prewritten text. The question-answering component infers answers using a hand-built knowledge-base about the domain, and converts them into English using simple template techniques before presenting them to the user. The resulting integrated system can be thought of as a "dynamic book" or "dynamic manual", where the user can not only browse and search prewritten material, but also proactively ask questions about what he/she is reading and receive answers back. These answers may be customized to the user's specific situation, and to the user's desired level

of detail; they also include answers requiring reasoning by the system.

In this paper, we present an implemented architecture for this style of question-answering system, which we have evolved over several years and several projects. The architecture includes both a pragmatic style of user interface, in which a user can both browse pre-written text and ask questions to the knowledge-base in an integrated fashion, and a reasoning component, for inferring situation-specific answers at run time. This approach evolved from our earlier work on a large-scale knowledge base about botany (Porter *et al.* 1988), and has been explored in several new domains including a distributing computing help-desk application (Clark & Porter 1996), and a training tool for performing a space science experiment (Clark, Thompson, & Dittmar 1998). In this latter application, the system included a simulation capability allowing the user (in simulation) to perform the experiment, and ask questions in context during the simulation.

We first discuss the motivation for knowledge-based question answering, and then provide an overview of the system architecture. We describe in detail how questions are answered in a four-step process of: creating an initial representation of the user's scenario of interest; inferring additional facts about it required for the answer, guided by an answer schema; assembling those facts into a single unit; and presenting that unit back to the user. Finally we critique the approach, highlighting both its strengths and weaknesses.

Motivation for Knowledge-Based Question-Answering

It is clear from the literature that for certain classes of questions, "simply" retrieving and reciting prewritten paragraphs of text can achieve high performance levels (e.g. START (Katz 1997), the Loebner "Turing test" winners (Mauldin 1994), modern case-based help-desk applications, Ask Jeeves (at ask.com)), even though such systems can be easily tricked, and rely heavily on the user to interpret and answer the question him/herself from the recited text. A knowledge-based question-answering system should thus be viewed as an

augmentation of, rather than a rival to, these retrieval-based approaches, to help with answering questions requiring situation-specific answers, where multiple pieces of information need to be combined, and where suitable pre-written text is not available. The potential benefits which the knowledge-based component can offer are as follows:

Customization: As answers are synthesized from a knowledge-base, answers can be customized to the user's particular situation, for example answering questions such as (from the space experiment application): "What is the state of the motor?", "Why did light 3 come on?", "What should I do next?".

Controllable level of detail: Similarly, the level of detail can be dynamically controlled to suit the user's level of expertise, by controlling how much information from the knowledge-base is included in the answer.

Robustness: By inferring answers at run-time, rather than reciting a pre-written paragraph of text, the system can respond to questions unanticipated at the time of system construction.

The prototypes described here demonstrate the first two of these benefits, in particular exploiting the use of inference over a scenario to a greater extent than earlier knowledge-based question-answering systems, such as Quest (Graesser & Franklin 1990), and by McKeown (McKeown 1985). We also believe they are on track towards achieving the third, the current robustness being limited by the incompleteness of the knowledge-bases constructed. We discuss our progress in achieving these goals in more detail after presenting the approach.

Approach and System Architecture

We first describe the overall architecture for our system, and then describe how the knowledge-base, inference engine, and user interface interact to provide run-time question-answering capabilities. We will mainly use the Distributed Computing Environment (DCE) application for illustration. Briefly, the goal of the DCE application was to augment on-line documentation about DCE with knowledge-based question-answering capabilities, to provide additional support for novice users trying to understand DCE. The examples thus include manual pages and questions about distributed computing concepts.

The architecture for the system is depicted in Figure 1, and a screendump from the Distributed Computing application is shown in Figure 2. As illustrated in these figures, the user interface consists of two side-by-side windows, the left-hand window displaying (for the DCE application) pre-written manual pages, and the right-hand window displaying a run-time-generated screen about a "topic", where a topic is a concept represented in the knowledge-base.

The pages in the left-hand window can be browsed by the user in the normal way, independent of any

question-answering facilities, by clicking on hyperlinks to take him/her to a new page. However, before displaying a page, the Hyperlinker module (Figure 1) first scans it for words or phrases denoting concepts which the system "knows about", i.e. which are represented in the knowledge base (each concept in the KB has an associated set of English words/phrases used to refer to it, and the hyperlinker scans the document for these words/phrases). It then adds hyperlinks to those words/phrases, indicating that that the user can click on them to get more information.

If the user clicks on one of these concepts, then the Page Generator constructs a page of information about that concept, and displays it in the right-hand window. In Figure 2, the user is viewing a page of information about the concept **Binding-Event**. To synthesize a page, the Page Generator retrieves and formats several pieces of information about the concept from the knowledge-base:

- the generalizations (superclasses) of the concept
- the specializations (subclasses) of the concept
- a definition of the concept (a prewritten piece of text)
- A list of applicable questions for that concept
- A list of related concepts (if any)
- Pointers to other relevant manual pages (if any)

The most important of these items is the list of questions, as these are the entry points to knowledge-based question answering. In a similar way to (McKeown 1985), the system has a (small) collection of generic question types (e.g. "Describe an X?", "Parts of X?") which it knows how to answer, i.e. has a question-answering procedure defined for it. Each question type is associated with a concept in the KB, and the set of applicable questions for a given concept are those which are attached to that concept or any of its superclasses. Thus, for example, the question "The equipment needed for X?" is attached (in the space experiment application) to the concept **Activity**, and thus would appear in the list of questions about concepts which were **Activitys**, but not in the list of questions for (say) physical objects. In this way, the system selects and displays only those questions which "make sense" for particular concepts, based on their type. Some of the question types and their attachment in the taxonomy for the distributed computing application are shown in Figure 3.

If the user clicks on one of the questions in the right-hand window, this starts the question-answering procedure. Before computing an answer, however, the user is first prompted to describe or select a particular scenario, which will then be used by the system as the context in which to generate an answer, customized to that scenario. In the example in Figure 2, the user clicked on "Describe a binding event", and then (not shown) was prompted to name the "players" in the scenario he/she was interested in (here, the client = Netscape, the server = a Web server, the server's machine name

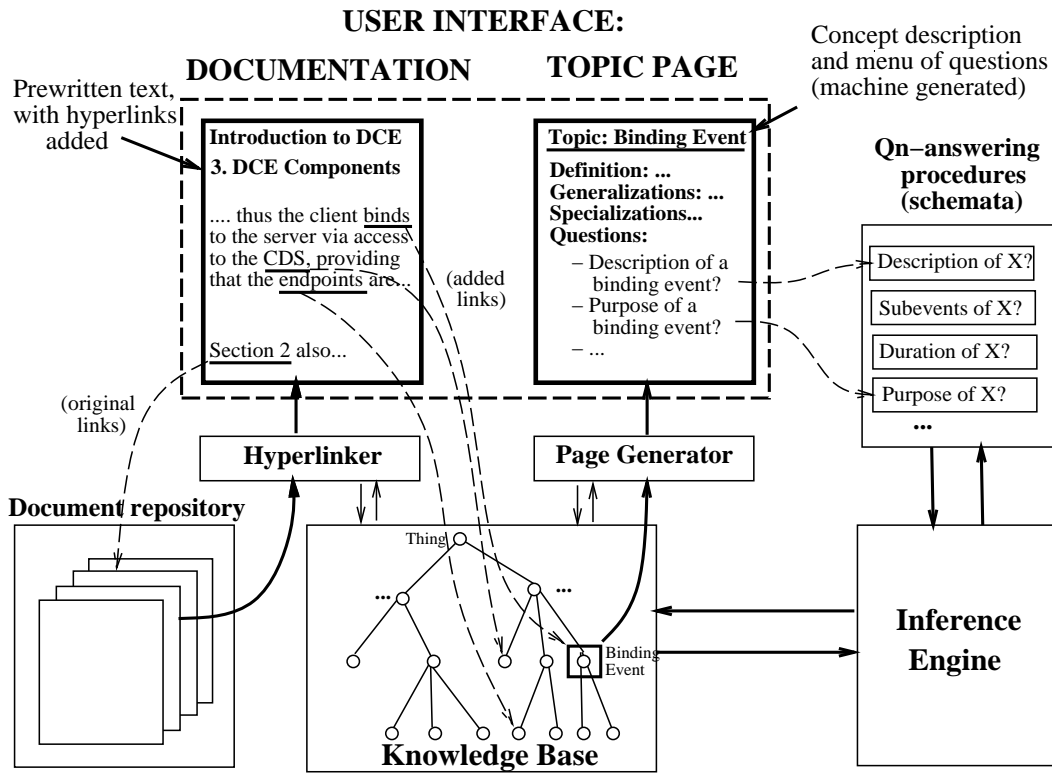


Figure 1: The overall application architecture. The left-hand window displays prewritten text, with hyperlinks to known concepts added. The right-hand window displays a machine-synthesized page about a selected topic, including hot-spots for asking questions about that topic, and their answers.

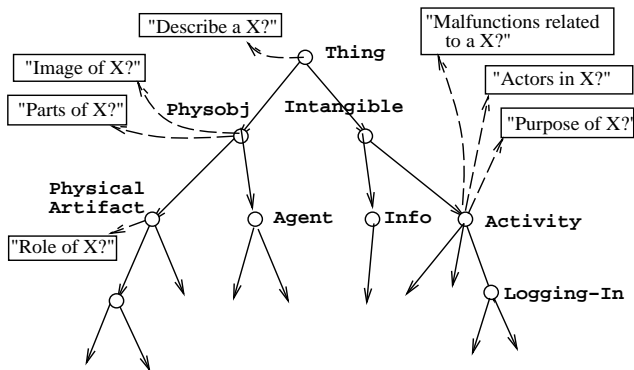


Figure 3: Different question types are attached to the most general concepts in the KB for which that question “makes sense”. This figure illustrates (most of) the question types for the distributed computing application.

= Zippy). Following this, the answer procedure was executed and an answer generated, customized to this scenario, and displayed back to the user (spliced into the original text in the window, under the question, as illustrated in Figure 2). Had the user provided a different scenario, a different answer would have been generated. As well as differing superficially in the names of the players involved, customized answers may also differ structurally, if the types of players selected affect the answer. For example, if an Oracle client binds in an idiosyncratic way, and the user had selected Oracle as the client when prompted for the scenario, then an entirely different answer reflecting this idiosyncrasy would have been generated for this question.

Having read the answer to his/her question, the user may ask other questions, click on other concepts to explore their information, or return to browsing the on-line material in the left-hand window. In this way, the user is able to navigate existing documentation, view information about concepts he/she reads about, ask questions and get customized, machine-generated responses about those concepts, and jump to related pages in the original text. This allows the user to mix passive browsing of information with active question-asking as suits his/her own learning style.

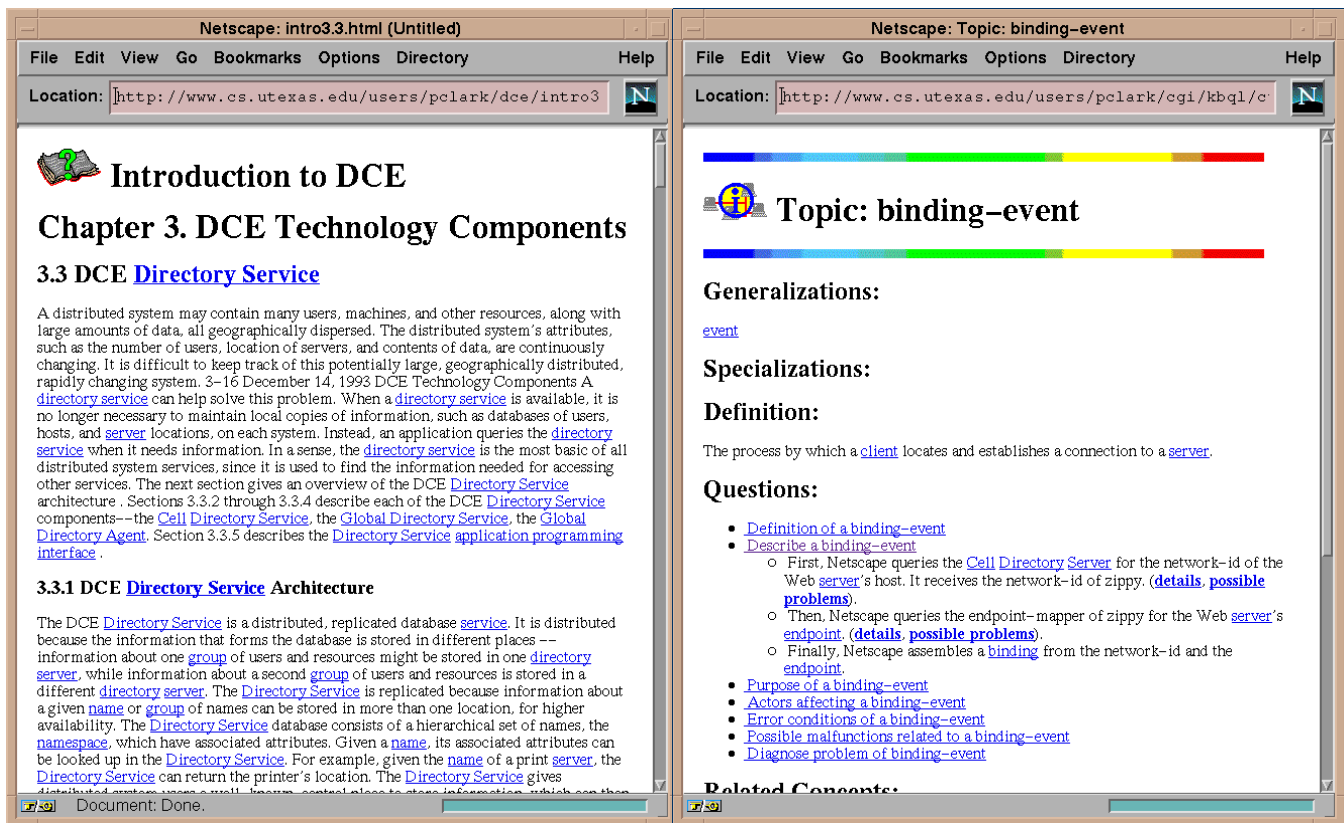


Figure 2: An example screen from the DCE application. The left-hand window displays pre-written text, while the right-hand window is machine-synthesized, displaying information about a user-selected concept (“topic”). Here, the user has clicked on the question “Describe a binding event”, and described the scenario he/she is interested in (not shown). The system has responded with a situation-specific, machine-synthesized answer as shown underneath this question, and also in Figure 4(i).

Question-Answering

Three examples of questions and answers which the system will generate, from different applications, are given in Figure 4 in text form (the actual interface is through a Web browser, as illustrated in Figure 2). Note that questions are answered by the system in the context of a particular scenario, part of which is assumed (based on the overall scope and purpose of the application), and part of which is specified by the user as described shortly. Thus the answers will vary, depending on the scenario which the user describes.

To pose a question about a concept which he/she is viewing (in the right-hand window, e.g. *Binding-Event* in Figure 2), the user first clicks on the appropriate question from the displayed list of applicable questions for that concept. The system will then request a scenario and synthesize an answer in four stages:

1. **Creation:** Ask for and create (a representation of) the initial scenario, to act as the context for answering the question in.
2. **Elaboration:** Using rules in the KB, infer addi-

tional information about the scenario which is needed for the answer. This step is guided by an “answer schema” for the question type, specifying what information should be included in the answer.

3. **Assembly:** Gather the inferred information together, and format it into a single (HTML) unit.
4. **Presentation:** Present this answer to the user.

We now describe each of these stages in detail.

1. Creating an Initial Scenario

As well as containing rules about domain concepts, the KB also contains a “working memory” to reason about particular scenarios the user may be interested in. When answering a question, the system populates this working memory with ground facts about instances of interest. These facts constitute a (partial) model of the user’s scenario, and can be thought of pictorially as an “instance graph”. In the subsequent step of elaboration, rules from the KB will be applied to these instances to infer additional properties of them, thus “elaborating” this graph. This organization of the knowledge-base is illustrated in Figure 5.

Question: Describe a binding event?
 where (user-selected scenario):
 Client: "Payday"
 Server: Oracle Sybase Web
 Server's host: "Speedy"

Answer (machine-synthesized):
 * First, Payday queries the cell directory server for the network id of Oracle.
 * Then Payday queries the endpoint-point mapper of Speedy for Oracle's endpoint.
 * Finally, Payday assembles a binding from the network-id and the endpoint.

(i) from the distributed computing domain

Question: How do I rework a rivet?
 where (user-selected scenario):
 Rivet type? R5GF R5FT R5FV
 Rivet diameter? $\frac{5}{32}$ " $\frac{3}{16}$ " $\frac{7}{32}$ "
 Skin thickness? .056 .072 $\geq .072$

Answer (machine-synthesized):
 The skin is thick enough for standard replacement:
 * Set the counterbore to diam 0.2585.
 * Apply the counterbore tool.
 * Insert a $\frac{7}{32}$ " rivet.
 * Rivet in place.

(ii) from the factory task domain

Question: Equipment needed for PHaSE step A2?
 where (scenario):
 (the current state of the simulation)

Answer (machine-synthesized):
 You will need:
 * the screwdriver with a 4.5mm blade.

(iii) from the space experiment domain

Figure 4: Examples of answers generated by the system. The user selects the question from a list of applicable question types, then is prompted to provide details of their scenario of interest as shown. Following this, the system synthesizes an answer using the knowledge-base. The interaction is shown here in plain text form (the actual interface is though a Web browser).

First the system seeds the instance graph with some initial instances representing the scenario of interest. These instances can be thought of as the "key players" in the scenario. These "key players", and their range of possible values, are listed on (the frame for) the concept which the user is currently looking at. In this way, the system knows which players it needs to ask the user to name/select. The system asks the user for the identity of these "players" by popping up a (HTML)

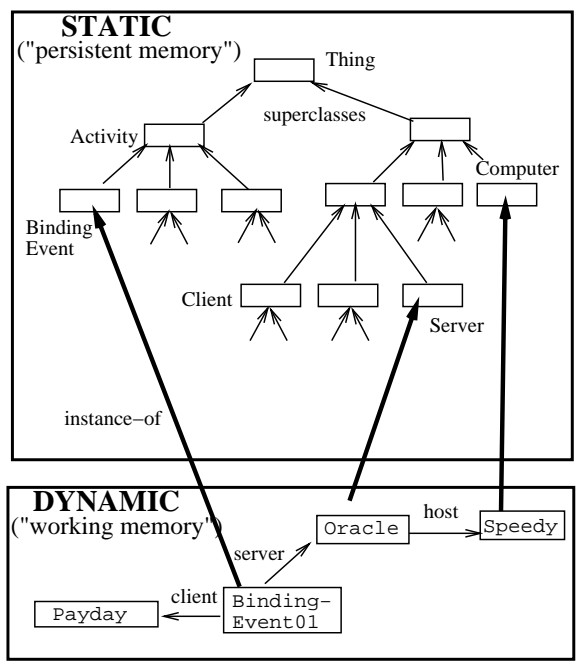
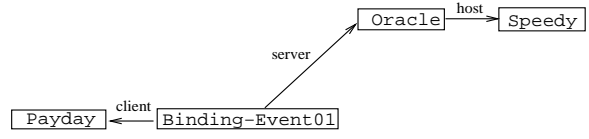


Figure 5: The knowledge-base contains two parts: The static, persistent part, containing frames representing domain knowledge, and the dynamic part, containing run-time generated instances which model a particular scenario of interest. The inference engine applies rules from the static part to infer properties of instances, thus elaborating the instance graph, in order to answer questions.

fill-out form for him/her to fill in (containing menus, radio-buttons, and text boxes, to select from as appropriate). Textual equivalents of these forms are shown in Figure 4. For example, the three "players" listed (in the KB) for Binding-Event are the client, the server, and the server's host machine: thus, after clicking on a question about binding events, the user is asked to select or provide values for these three players.

Following this, the system creates instances denoting these players in the working memory. For the "Describe a binding event?" question in Figure 4, the working memory will contain the initial "instance graph" as shown below:



2. Schema-Guided Elaboration (Inference)

The second stage of question-answering is to look up or infer the various pieces of information required in the answer. To do this, the system must first know what constitutes an answer to a question, i.e. what information is required in the answer, and in what order it should be presented. This is specified using an answer

<p>“Describe event X?”</p> <p>For each subevent SE of event X [Query1]:</p> <ol style="list-style-type: none"> 1. Present a summary of SE [Query2]. 2. If there are recursively more subevents [Query3], add a “see details” hot-spot hyperlinked to (the procedure for) “Describe event SE?”. 3. If there are any known problems which can occur [Query4], add a “possible problems” hot-spot hyperlinked to (the procedure for) “Possible problems of SE?”. <p>“Parts of X?”</p> <p>Foreach part P of X [Query1]:</p> <ol style="list-style-type: none"> 1. Present the name of P [Query2] 2. If P itself has subparts [Query3], add a “see subparts” hot-spot hyperlinked to (the procedure for) “Parts of P?” <p>“Image of object X?”</p> <p>Find any image file associated with X [Query1]. Display that image.</p>
--

Figure 6: Answer schemata for three question types. Each schema specifies what an answer should contain, and how the components should be assembled together. Each component has a KB query associated with it, denoted [QueryN], which is sent to the KB to compute its value when it is needed.

schema (what we have been referring to so far as an “answer procedure”) for the type of question being asked, each question type having a single answer schema associated with it¹. The answer schema specifies which components of information are required in an answer, and how they should be presented. Each of these components has a KB query associated with it, which can be submitted to the inference engine to find that information in the context of the current scenario. Figure 6 shows some examples of answer schemata. This approach is similar to (and inspired by) the use of Explanation Design Packages in the Knight system (Lester & Porter 1997), and also similar in style to the use of schemata in other question-answering systems, e.g. (McCoy 1989), (McKeown 1985).

Inference: When a schema is to be filled in, the queries it contains are sent to the inference engine, and the answers computed and returned to the schema. To answer these queries, the inference engine may need to apply general rules from the KB to the instance graph

¹In fact, a single question type may sometimes have multiple answer schemata, each handling a different class of object. For example, “Describe a X?” has two answer schemata, one for when X is an object, and one for when X is an activity. The question type and object type will always uniquely determine the appropriate schema to use.

to infer the requested properties.

The KB itself is expressed using the frame-based representation language KM (Clark & Porter 1998b), similar in style to KRL (Bobrow & Winograd 1985). Each concept in the domain is represented by a frame, with slots and values. Values are themselves either frames, or expressions which evaluate to (“point to”) other frames. Formally, this is just a convenient notation for Horn clause rules in typed first-order logic, where frame names correspond to classes (types) or instances (constants), slots correspond to relations, and frame structures correspond to a grouping of Horn clause rules which share the same type restriction for their head predicate’s first argument. (Part of) the frame for the concept **Binding-Event** in the distributed computing application is shown in Figure 7. Full details of this KR language are given in (Clark & Porter 1998b).

Each query in the schema asks for a property of an instance in the scenario. For example, [Query1] in the “Describe event X?” schema (Figure 6) is the query:

(the subevents of X)

where X is the (instance of the) topic concept in the scenario (e.g. **Binding-Event01**). The inference engine finds that property, either by a simple lookup in the instance graph, or (if is not present) by evaluating an expression for it, inherited from one of that instance’s classes in the KB. Evaluating the expression may involve recursive calls of the inference engine. The results are added to the instance graph, and thus inference can be thought of as “growing” the instance graph in a lazy fashion by applying rules in the KB, thereby elaborating the graph just enough to answer the queries posed to it by the answer schema. Typically, different initial instances will result in the graph growing in different ways, depending on their properties.

Some exchanges with the inference engine are shown below, here issued manually by the knowledge engineer at the interpreter prompt (“KM>”). During question-answering, queries like these are instead issued automatically by the answer schema (comments are preceded by “;”):

```

;;; "Create an instance graph representing the
;;; initial scenario (a binding event with client,
;;; server, and server's host specified)."
KM> (a Binding-Event with          ; <= user's input
      (client (Payday))
      (server ((a Oracle-Server with
                 (host (Speedy))))))
      (Binding-Event01)          ; <= system's response

;;; "What are the subevents of this binding event?"
KM> (the subevents of Binding-Event01) ; <= user
      (Query01 Query02 Assemble01)    ; <= response

;;; "Who is the agent in the second subevent?"
;;; [Result = inherit and evaluate the path
;;;      '(the client of Self)' from Binding-Event])
KM> (the agent of Query02)
      (Payday)

```

```

(Binding-Event has (superclasses (Event))) ; [1]

(every Binding-Event has
 (client ((a Client-Process))) ; [2]
 (server ((a Server-Process)))
 (network ((a Network)))
 (subevents (
 (a Query with ; [3]
 (agent ((the client of Self))) ; [4]
 (queried ((the cell-directory-server of
 (the network of Self)))) ; [5]
 (requested ((the network-id of (the host of
 (the server of Self))))))
 (a Query with
 (agent ((the client of Self)))
 (queried ((the endpoint-mapper of
 (the host of
 (the server of Self))))))
 (requested ((the endpoint of
 (the server of Self))))))
 (a Assemble with
 (agent ((the client of Self)))
 (assembled ((a Binding)))
 (components (
 (the network-id of
 (the host of (the server of Self))))))
 (players ((the client of Self)
 (the server of Self)
 (the host of (the server of Self))))))

```

Semantics of selected expressions:

- [1] $superclasses(Binding-Event, Event)$.
- [2] $\forall b isa(b, Binding-Event) \rightarrow \exists c isa(c, Client-Process) \wedge client(b, c)$.
- [3] $\forall b isa(b, Binding-Event) \rightarrow (\exists q isa(q, Query) \wedge$
- [4] $(\forall c client(b, c) \rightarrow agent(q, c)) \wedge$
- [5] $(\forall q, n network(b, n) \wedge cell-directory-server(n, d) \rightarrow queried(q, d)) \wedge$
(etc)

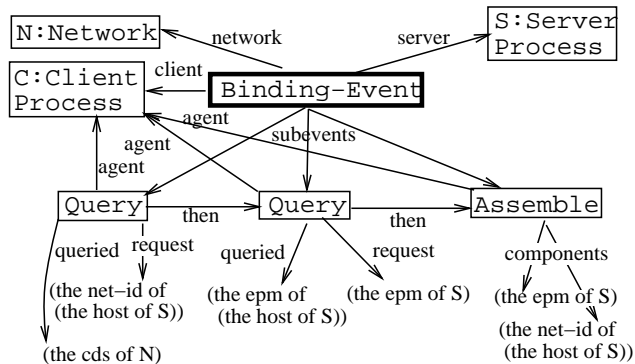


Figure 7: Representation of the concept Binding-Event in the distributed computing KB, shown both formally in the language KM, and informally sketched as a graph. Note the access paths (the X of (the Y of ...)), which are resolved at run-time by the inference engine for the particular instance of Binding-Event (denoted by "Self", above) under consideration.

```

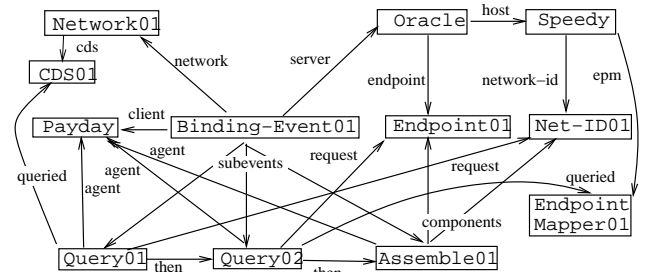
;;; "Who is queried in the second subevent?"
KM> (the queried of Query02)
(Endpoint-Mapper01)

;;; "Which endpoint mapper is this?"
KM> (the endpoint-mapper-for of Endpoint-Mapper02)
(Speedy)

```

Note that to answer these queries, the inference engine has pulled in information from the Binding-Event frame (and other frames) in the KB. For example, the subevents of Binding-Event01 are found by evaluating the expression from the subevents slot on Binding-Event (Figure 7), creating the three instances (Query01 Query02 Assemble01) denoting the three subevents.

Queries such as the ones above are attached to the answer schemata, and the system issues them at run-time to find the components for an answer. As a result of each query, the instance graph is gradually elaborated. After the inference engine has answered these queries, the graph contains:



The additional nodes and arcs depict the additional information inferred about the binding event (and its related instances) during the inference process, when answering the queries.

Text Generation: Finally, English sentences are generated from this graph by again sending queries to the inference engine. For this mechanism, "fill in the blank" text templates are placed on the text slots of frames, and queries for those slots' values cause the inference engine to instantiate them. For example, the frame for Query includes:

```

(every Query has
 (text ((the agent of Self) "queries"
 (the queried of Self) "for"
 (the request of Self))))

```

denoting the template:

```

<agent> "queries" <queried> "for" <request> "."

```

By querying for this slot on a Query instance, the template is inherited, the paths it contains evaluated (thus instantiating the template with appropriate instances in the current instance graph), and the result returned. The answer schemata use queries like this to convert raw KB answers to English text for presentation to the user. A simple "make sentence" procedure then replaces instances with their print-names, concatenates

the result, capitalizes the first letter, and adds a period. We illustrate this text generation process:

```
; Create an instance of Joe querying Fred...
KM> (a Query with
      (agent (Fred))
      (queried (Joe))
      (request ((a Address))))
Query03

; Describe Query03. [Result=instantiated template]
KM> (the text of Query03)
(Fred "queries" Joe "for" Address01)

; Concatenate these pieces...
KM> (make_sentence (the text of Query03))
"Fred queries joe for the address."

; Same for Query02 earlier...
KM> (make_sentence (the text of Query02))
"Payroll queries the endpoint-mapper of speedy for
the oracle-server's endpoint."

; Create a binding-event with no customization..
KM> (the subevents of (a Binding-Event))
(Query04 Query05 Assemble06)

; and describe its query subevent...
KM> (make_sentence (the text of Query05))
"The client-process queries the endpoint-mapper of
the server-machine for the server-process endpoint"
```

These examples all show the same template being filled in in different ways. In the last example, as we did not specify the client for this new binding event, all KM knows is that it is a `Client-Process` (from the `client` slot on `Binding-Event` in Figure 7), hence the name “the client-process” is constructed (similarly for other fillers).

3. Assembling the Answer

As well as specifying which components are needed to answer a question, the answer schemata specify how to assemble the components into a single structure, as shown in Figure 6. This involves straightforward manipulation of the answers from the queries, for example concatenating strings and adding low-level HTML formatting commands (e.g. bullets, extra hyperlinks) so that the result is displayable through the Web interface.

4. Presentation to the User

Finally, the answer is relayed back to the user, and displayed. To do this, a separate module keeps track of the right-hand page the user is currently viewing, and constructs a new page, similar to the one the user is viewing except with the assembled answer spliced in. This new page is then returned to the user, giving the user the impression the page has been “expanded” to include the answer, as illustrated in Figure 2. From here, the user can click on hyperlinks to get more details, view different concepts, or return to browsing the original material in the left-hand window.

Status, Discussion, and Lessons Learned

Although we have not performed a formal evaluation of this approach, there are a substantial number of significant and interesting issues which we now discuss.

This architecture was originally developed for the distributed computing “help-desk” prototype (Clark & Porter 1996). Since then, it has also been applied to an aircraft assembly task at Boeing, and for a knowledge-based tutoring system for a space experiment (Clark, Thompson, & Dittmar 1998). In addition, simple illustrations of the approach have been built for two other domains. These prototypes have been well-received, but none have achieved regular usage primarily due to the incompleteness of the underlying knowledge-bases, as we discuss shortly.

The two most significant potential benefits which the knowledge-based approach provides are the ability to generate situation-specific answers at run-time, and to answer questions unanticipated at the time of system construction. We have described how the former of these has been achieved, through the use of run-time inference to model and reason about a user-provided scenario. Such situation-specific questions are difficult or impossible to answer if only pre-written paragraphs of text are available. This was a major motivation for the original distributed computing application, where the typically general-worded text of on-line documentation is often unhelpful to users, who frequently have difficulty working out what applies to their situation.

The ability to answer unanticipated questions is based on the idea that general knowledge in the KB (e.g. for the space application, that lights can burn out) can be applied to specific concepts (e.g. the power-on light in the space experiment) at run-time. We were able to demonstrate this capability in a limited way, for specific questions, but to achieve this more fully requires that the knowledge-base be more complete, with substantially more knowledge about the general concepts in the KB.

The main bottleneck in these projects, as is well-known in the literature, is the task of building the knowledge-base in the first place. For all these applications, only a part of the knowledge required to cover the application domain was encoded, and the final systems were only able to answer a handful of questions; substantially more effort would have been needed for complete coverage. This barrier has been a stumbling-block for many knowledge-based projects; the ‘solution’, which we are actively pursuing as a research theme, is the construction of a knowledge-base from prefabricated, reusable knowledge components, rather than hand-built from scratch (Clark & Porter 1997). This approach is still a research topic, but we are hopeful that it can significantly ease the knowledge modeling bottleneck in the future.

A second, related issue is that constructing a knowledge-base requires a skilled knowledge engineer.

We thus also see the need for powerful but easy-to-use authoring tools that might allow a subject matter expert (who is not also a knowledge engineer) to enter at least some of the required knowledge via a simple interface, or using ‘mediating representations’ (Boose 1990). The lack of easy-to-use authoring tools was the primary reason that the factory task application was not pursued beyond a simple demonstration prototype.

Another limitation of the original DCE application was the inability of the knowledge-base to model dynamic situations, i.e. to represent and reason about actions and their effects. As a result, certain important classes of questions requiring hypothetical reasoning (“What if...?”, “Why is ...?”²), could not be handled by the DCE application. A major step forward was to add a simulation capability to the knowledge representation language KM in 1998 (based on situation calculus), so that it could model and reason about world dynamics (Clark & Porter 1998a). This was used for the space application (Clark, Thompson, & Dittmar 1998), where the user could not only browse text, but also perform the experiment in simulation (by selecting actions to perform at each step), and pose questions in the context of the current simulation state. This also improved on the original DCE application as the scenario for the question was then presented automatically (i.e. as the current simulation state), rather than having to be manually entered by the user every time a question was asked.

Of course, not all questions that a user has are situation-specific; for many, simply retrieving a paragraph of pre-written text would suffice. Although the user can browse and access pre-written text in our architecture, we have not added capabilities to better search or index such text, and instead have focussed primarily on the knowledge-based aspect. It would clearly be beneficial for future applications to better incorporate text-retrieval methods, for questions where customization is not required.

In addition, there are certain classes of questions which are outside the scope of this framework, particularly those which require a sustained dialog (e.g. diagnosis) rather than a “one-shot” question-and-response. Dealing with these types of questions is even more challenging, but also has high payback as this style of question-answering is even further outside the scope of text-recital approaches.

Despite the cost of a knowledge-based approach, we believe that in the long-run a question-answering system should at least “understand” (i.e. have an inference-capable internal representation of) what it is talking about, if it is to avoid brittleness and achieve the versatility that people ultimately desire. The architecture here presents an implemented, practical framework for such systems, and illustrates some of the potential benefits which can thus be provided.

²requiring reasoning about the hypothetical situation “What if it wasn’t?”

References

- Bobrow, D., and Winograd, T. 1985. An overview of KRL, a knowledge representation language. In Brachman, R., and Levesque, H., eds., *Readings in Knowledge Representation*. CA: Kaufmann. 264–285. (originally in *Cognitive Science* 1 (1), 1977, 3–46).
- Boose, J. 1990. Knowledge acquisition tools, methods, and mediating representations. In *Proceedings of the First Japanese Knowledge Acquisition for Knowledge-Based Systems Workshop (JKAW’90)*.
- Clark, P., and Porter, B. 1996. The DCE help-desk project. (www.cs.utexas.edu/users/mfkb/dce.html).
- Clark, P., and Porter, B. 1997. Building concept representations from reusable components. In *AAAI-97*, 369–376. CA: AAAI. (<http://www.cs.utexas.edu/users/pclark/papers>).
- Clark, P., and Porter, B. 1998a. KM – situations, simulations, and possible worlds. Technical report, AI Lab, Univ Texas at Austin. (<http://www.cs.utexas.edu/users/mfkb/km.html>).
- Clark, P., and Porter, B. 1998b. KM – the knowledge machine: Users manual. Technical report, AI Lab, Univ Texas at Austin. (<http://www.cs.utexas.edu/users/mfkb/km.html>).
- Clark, P.; Thompson, J.; and Dittmar, M. 1998. KB-PHaSE: A knowledge-based training tool for a space station experiment. Technical Report SSGTECH-98-035, Boeing Applied Research and Technology, Seattle, WA. (<http://www.cs.utexas.edu/users/pclark/papers>).
- Graesser, A. C., and Franklin, S. P. 1990. QUEST: A cognitive model of question answering. *Discourse Processes* 13:279–303.
- Katz, B. 1997. From sentence processing to information access on the world-wide web. In *Proc. AAAI Spring Symposium on Natural Language Processing for the World Wide Web*. CA: AAAI. (<http://www.ai.mit.edu/projects/infolab/>).
- Lester, J. C., and Porter, B. W. 1997. Developing and empirically evaluating robust explanation generators: The knight experiments. *Computational Linguistics* 23(1):65–101.
- Mauldin, M. L. 1994. Chatterbots, tinymuds, and the turing test: Entering the loebner prize competition. In *AAAI-94*, 16–21. CA: AAAI. (also see <http://www.loebner.net/Prize/loebner-prize.html>).
- McCoy, K. F. 1989. Generating context-sensitive responses to object-related misconceptions. *Artificial Intelligence* 41(2):157–196.
- McKeown, K. R. 1985. *Text Generation*. UK: Cambridge Univ. Press.
- Porter, B. W.; Lester, J.; Murray, K.; Pittman, K.; Souther, A.; Acker, L.; and Jones, T. 1988. AI research in the context of a multifunctional knowledge base: The botany knowledge base project. Tech Report AI-88-88, Dept CS, Univ Texas at Austin.