Knowledge Entry as the Graphical Assembly of Components

Peter Clark
John Thompson
Knowledge Systems
M&CT, Boeing
Seattle, WA 98124

Ken Barker
Bruce Porter
Computer Science
Univ. Texas
Austin, TX 78712

Vinay Chaudhri
Andres Rodriguez
Jerome Thomere
Sunil Mishra
SRI International, CA 94025

Yolanda Gil
ISI
USC
Marina del Rey
CA 90292

Pat Hayes
Thomas Reich-herzer
IHMC, U W Florida
FL 32514

Abstract

Despite some successes, the lack of tools to allow subject matter experts to directly enter, query, and debug formal domain knowledge in a knowledge-base still remains a major obstacle to their deployment. Our goal is to create such tools, so that a trained knowledge engineer is no longer required to mediate the interaction. This paper presents our work on the knowledge entry part of this overall knowledge capture task, which is based on several claims: that users can construct representations by connecting pre-fabricated, representational components, rather than writing low-level axioms; that these components can be presented to users as graphs; and the user can then perform composition through graph manipulation operations. To operationalize this, we have developed a novel technique of graphical dialog using *examples* of the component concepts, followed by an automated process for generalizing the user's graphically-entered assertions into axioms. We present these claims, our approach, the system (called SHAKEN) that we are developing, and an evaluation of our progress based on having users encode knowledge using the system.

Keywords

Graphical knowledge entry, knowledge acquisition, components, composition, knowledge-based systems.

INTRODUCTION

Despite some successes, the lack of tools to allow subject matter experts to directly enter, query, and debug formal domain knowledge in a knowledge-base (KB) still remains a major obstacle to their deployment. Our goal is to create such tools, so that a trained knowledge engineer is no longer required to mediate the interaction. This paper presents our work on the knowledge entry part of this overall knowledge capture task. In particular, we present a novel technique of graphical dialog using *examples* of component concepts, and an evaluation of this technique. The particular application domain we are working with is cell biology (although our techniques are not specific to this domain), and our focus has been on capturing domain knowledge, as opposed to problem-solving knowledge (the "what" rather than the "how to" knowledge). This work is being conducted as part of DARPA's Rapid Knowledge Formation (RKF) project [8].

K-CAP'01, October 22-23, 2001, Victoria, British Columbia, Canada.

CONTEXT, GOALS, AND CLAIMS

Context of the Work

Component-based approaches for knowledge capture are currently popular, and so we first describe where our work fits in this context. It is useful to view expertise as comprising *problem-solving* knowledge ("how" knowledge) and *domain* knowledge ("what" knowledge). Perhaps the most successful component-based, knowledge capture work has been with problem-solving knowledge, where reusable problem-solving methods (PSMs) can be assembled to produce task-specific problem-solvers, e.g., [4, 18]. Moreover, PSMs can also be used to guide acquisition of domain facts, as they "expect" certain types of knowledge in order to operate, e.g., the Expect system [2], Protege-derived tools [13].

Less work has been devoted to capture of domain knowledge, and it is capture of this kind of knowledge that is the primary focus of our work. Existing tools have focussed only on entry of taxonomic ("isa") knowledge and database-style facts, e.g., WebOnto [9], or have been targeted for use by knowledge engineers rather than subject matter experts, requiring logical axioms to be directly entered, e.g., Ontolingua [10], GKB [15], and the HITS Knowledge Editor [17]. Our goal is to allow users to encode these more complex, declarative axioms, describing both static objects and dynamic processes in the world, without requiring expertise in logic or AI. Our approach is analagous to work on PSMs: We similarly assume a library of components (but about objects and processes in the world, rather than about problem-solving strategies); and components similarly provide "expectations" to guide the user (but about how domain knowledge should be represented). The result of the knowledge capture process is a new set of axioms about a domain-specific object or process, which can then be used for question-answering.

Concerning our use of graphs for interacting with users, graph-

ical notations have frequently been found to be intuitive to users, e.g., in "concept maps," an informal graphical notation developed for educational settings, and used in tools such as WebMap [12] and by Univ. West Florida [3]. Similarly they have sometimes been found intuitive to knowledge engineers themselves, e.g., [16, 11]. Our goal is to exploit this intuitiveness for working with a pre-built library of representational components.

Claims

A central claim of our approach is that users can construct axiomatic representations by connecting pre-fabricated, representational components, rather than writing low-level axioms directly. By component, we mean a coherent set of axioms which describe some abstract phenomenon (e.g. the concept of "invade"), and which are presented to the users as a single representational unit. By composition, we mean the connection of such components together, and the computation of additional implications of the composite set of axioms. Components are intended to encode fairly abstract phenomena, such as knowledge about the concepts "invade," "break," "container," and "control system." Our goal is thus to recast the knowledge capture process as one of instantiation and assembly, rather than of axiom writing.

A second, related claim of our work is that components can be presented to users as graphs, and the user can then perform composition through graph manipulation operations. As a result, details of the underlying logic are hidden from users. Two implementation challenges for this are first expressing components as graphs, and second translating the user's graph manipulation operations back into logic, so that as the user manipulates graphs, the system records the logical equivalent of those operations. Our novel solution to this challenge is to have the dialog with the users be in terms of *examples* of their concepts of interest, coupled with a process for generalizing the user's graphically-entered assertions.

These two claims are related. In particular, the claim that knowledge capture can be treated primarily as an assembly process suggests that just a small number of axiom types (for stating connections and instantiations of existing components) will be sufficient to allow the users to build adequate representations. Although any full axiomatization of the user's concepts of interest may require complex axioms about space, time, actions, movement, etc., these will have been pre-built in the KB, and the user's job is thus simplified to describing domain-specific concepts using them. This provides a basis for the design of the graphical interface, as it only needs to support entry of this restricted set of axiom types, rather than the full range of possible first-order logic expressions. To the extent that these claims hold, a parsimonious tool for knowledge capture can be constructed, and to the extent that they do not, special add-ons will be needed to accomodate the user's needs.

TECHNICAL APPROACH

The user's goal is to create/extend a representation of a concept, i.e., encode axioms describing the properties, structure, and behavior of some domain-specific object or process. The user's activities are to: (1) Identify relevant components from the prebuilt KB; (2) connect and extend them to build a new representation; (3) save the result; and (4) test and ask questions about the new concept. The focus of this paper is on step 2 above, the construction of new representations.

Components

A component is a small set of first-order logic (FOL) axioms about a particular concept, gathered into a single data structure, encoding a coherent description of that concept [5]. The user is provided with a pre-built library of such components to work with. (Creating this library is a separate, major goal of our project [1]). For example, consider a (much simplified) component describing the process of "Invasion". It might include axioms stating that:

- The defending object has some barrier to protect it
- During an invasion, the invader penetrates that defensive barrier, then enters through it, then takes control of the attacked object.
- The invading agent is a tangible entity
- etc.

Statements such of these are encoded in first-order logic in the KB using (in our case) the frame-based language KM [6]. A simplified example of this notion of invade looks (in KM notation, with examples of equivalent FOL notation as footnotes)[1]:

```
;;; [1] "The invading agent is a tangible entity"
;;; [2] "The subevents of an invasion are a penetrate,
;;;       an enter, and a take control."
;;; [3] "During the penetrate, the invader penetrates
;;;     the defensive barrier of the attacked object."
;;; [4] "The first subevent is a penetrate event"
;;; etc.
(Invade has (superclasses (Attack)))
(every Invade has
 (agent ((a Tangible-Entity)))          ;[1]
 (object ((a Tangible-Entity with
            (has-part ((a Barrier))))))
 (subevent (
   (a Penetrate with                    ;[2a]
    (agent ((the agent of Self)))       ;[3a]
    (object ((the Barrier has-part of ;[3b]
                (the object of Self))))
    (next-event ((the Enter subevent
                           of Self))))
   (a Enter with                        ;[2b]
    (agent ((the agent of Self)))
    (object ((the object of Self)))
    (next-event (
      (the TakeControl subevent of Self))))
   (a TakeControl with                  ;[2c]
```

[1] Briefly on KM's syntax: slots (lowercase) are binary predicates, classes (mixed case) are sorts/types, '`every`' denotes universal quantification and '`a`' denotes existential quantification. See [6] for further details.

```
    (agent ((the agent of Self)))
    (object ((the object of Self))))))
 (first-subevent (
   (the Penetrate subevent of Self)))) ;[4]
```

Or in standard FOL syntax:

[1] $\forall i\ isa(i, Invade) \rightarrow$
$\exists y\ isa(y, Tangible\text{-}Entity) \wedge agent(i, y)$

[2a,3] $\forall i\ isa(i, Invade) \rightarrow$
$(\exists p\ subevent(i, p) \wedge isa(p, Penetrate) \wedge$
$(\forall a\ agent(i, a) \rightarrow agent(p, a)) \wedge$
$(\forall o, b\ object(i, a) \wedge has\text{-}part(o, b) \wedge$
$isa(b, Barrier) \rightarrow object(p, b)))$

etc.

These axioms provide one fairly general model of invasion for the user to start from, and use concepts which themselves already have rich semantics in the KB. For example, axioms about the concept of `Enter` (not shown here) encode that if something is entered, then the entering object will be spatially inside afterwards, that the path of entry will necessarily cross the boundary of the entered thing, etc. The KB uses a rich language for describing the properties and effects of actions, allowing questions to be answered through both deductive reasoning and running simulations.

Displaying Axioms to the User

To present the axioms about a concept C to the user, the raw axioms are not presented directly. Rather, the user sees an *example* I of that concept, i.e., a set of ground facts about I, computed from those axioms. Ground facts are both comprehensible and graphable, and provide an easy-to-grasp (although approximate) summary of what the KB "has to say" about a concept. The user then builds new concepts by interacting with this and other examples.

For instance, suppose the user is wanting to build a representation of how a virus invades a cell. One starting point for this is the pre-built concept of $Invade$, which the user would locate by browsing the component library. To then display the axioms for invade, our system SHAKEN then:

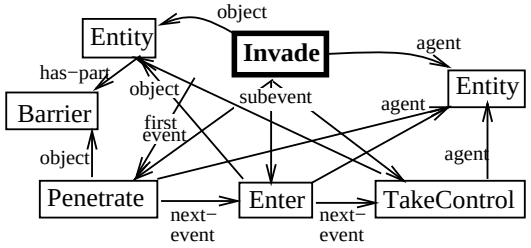
1. creates an instance I of $Invade$ (i.e., asserts the existence of an individual of type $Invade$), then
2. queries the KB for values for each of I's slots (i.e., uses inference to compute all ground facts of the form $slot(\mathrm{I}, x)$). Existentially quantified variables are Skolemized, and thus the result of this is typically a set of ground, binary facts between Skolem individuals. An example is shown shortly.
3. Recursively applies step 2 to each such value x found, up to a certain depth limit.
4. Presents this database of ground facts to the user as a graph, where each Skolem instance is a node, and each binary relation is an arc. Nodes are labelled with the most specific class(es) (i.e., sort, type) that each instance belongs to.

The boundaries of this procedure, and hence the extent of the resulting graph, are set manually by us, the knowledge engineers, by pre-specifying which slots should be included in the graph, and the depth of recursion. An autolayout algorithm then determines the spatial layout of nodes and arcs. The graphs are computed dynamically by this procedure at run-time, and thus can automatically adapt as new axioms are added to the system. From here, the user can modify the initial presentation by moving, expanding, or contracting nodes in the graph, hiding or exposing edges, and saving his/her revised presentation so new uses of that concept will appear the same way.

Applying this procedure to our (simplified) $Invade$ representation, SHAKEN would first generate a Skolem instance denoting an example of invade, e.g., named $Invade1$ (terms ending with numbers denote Skolem constants), and hence the set of facts:

agent(Invade1, Tangible-Entity2)
object(Invade1, Tangible-Entity3)
has-part(Tangible-Entity3, Barrier4)
first-subevent(Invade1, Penetrate5)
subevent(Invade1, Penetrate5)
agent(Penetrate5, Tangible-Entity2)
object(Penetrate5, Barrier4)
next-event(Penetrate5, Enter6)
subevent(Invade1, Enter6)
agent(Enter6, Tangible-Entity2)
object(Enter6, Tangible-Entity3)
next-event(Enter6, TakeControl7)
subevent(Invade1, TakeControl7)
agent(TakeControl7, Tangible-Entity2)
object(TakeControl7, Tangible-Entity3)

From this, a graph would be synthesized and displayed, where each node corresponds to a Skolem instance and each arc a binary relation. The resulting graph may look, for example:



Entering Knowledge by Interacting with the Graph

Suppose that the user wishes to build a representation of how a virus invades a cell. He/she would first provide a name for this new concept (e.g., $VirusInvasion$), and then locate one or more components in the KB to start from. In this example, the user may select the $Invade$ concept shown earlier. As a result, SHAKEN generates the above database, and also adds the assertion that the root instance denotes (an example of) the user's new concept, i.e., asserts:

$isa(Invade1, VirusInvasion)$

As a result, the label on this root node appears as "Virus-Invasion," and would appear as shown in the top graph in Figure 1. The entire graph is treated as a "representative instance" of the user's new concept.

To develop a model of how (for example) a virus invades a cell using this and other components, the user needs to encode facts such as:

A1. the invading agent is a virus
A2. the invaded object is a cell
A3. the penetrate is by means of endocytosis
A4. the agent in the endocytosis is the invaded object (i.e., the cell)
A5. there is also a delivery (of DNA) taking place
A6. there are certain correspondences between the invade and the delivery e.g.,
  A6.1 the invader (i.e., the virus) is the same as the agent in the delivery
  A6.2 the thing delivered is the DNA of that virus.

Rather than writing these statements in logic, the user makes them through the graphical interface via graph manipulation operations. This is possible because these axioms (specifying the composition) are generally all of a simple form: the complex axioms about virus invading a cell, e.g., how the spatial relationships of the objects change during the process, are mainly applications of more general axioms which already reside in more general components, and thus have already been pre-encoded in the component library. The user's job (and thus the interface) is thus simplified to using just this restricted subset of axiom types. As stated earlier, this is an important claim of our work, namely that by pre-encoding components well, a set of simple types of connections between them will be adequate for KB construction by a user who is not a trained knowledge engineer.

SHAKEN currently supports four types of axiom-building graph operations (plus others for controlling layout and node visibility). Each graphical operation corresponds to a simple, ground assertion about the example he/she is working on, and each time the user performs an operation, SHAKEN makes the corresponding logical assertion in the KB. On completion, an algorithm generalizes these assertions to hold for *all* instances of the concept C the user is describing, and the resulting axiom set captures the knowledge the user has entered. If the user is happy with his/her work, the axiom set is added to the KB, and can be further refined later and/or used itself as a component for defining new concepts.

The four graphical operations and their corresponding axioms are listed in Table 1 and illustrated in Figure 1. They are as follows:

Specialize: Refine an object's most specific class(es). In Figure 1, the user has clicked on the first $Entity$ node and then selected $Cell$ from a menu, to state that the thing being invaded is a cell. This asserts *isa(Tangible-Entity3,Cell)* in the KB.

Add: Add a new participant to the representation. In Figure 1, the user has selected the graph for $Virus$. This asserts $\exists x\ isa(x, Virus)$, which is then Skolemized to $isa(Virus8, Virus)$ and asserted in the KB.

Unify: State that two objects are coreferential. In Figure 1,

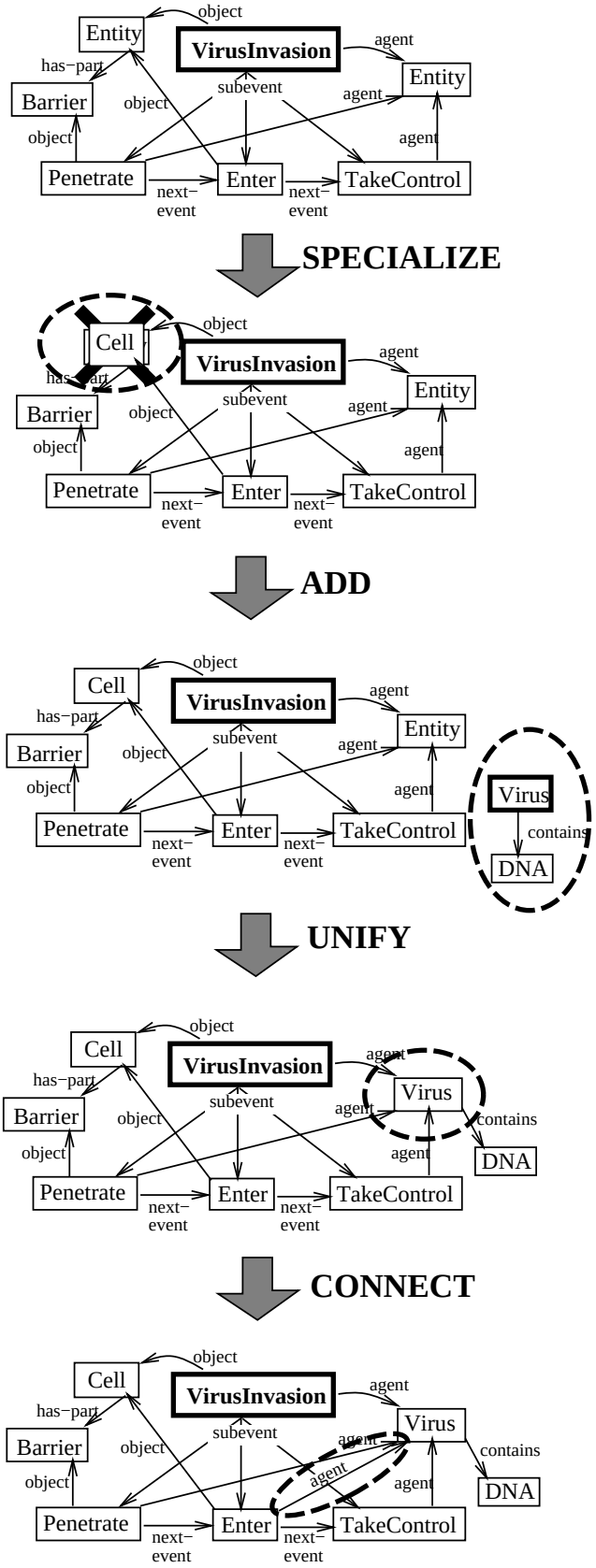


Figure 1: Examples of the four axiom-asserting graphical operations that the user can use in SHAKEN.

Operation	Examples	Graphical Action	Graphical Result	Logical Assertion
specialize	**A1**, **A2**	click node I + select class	I's label changes to $Class$	$isa(I, Class)$
add	**A3**, **A5**	click button + select class	graph for class appears	$\exists e\ isa(e, Class)$
unify	**A4**, **A6.1**, **A6.2**	drag node I onto node I'	nodes fuse	$I = I'$
connect	**A3**, **A4**	sketch arc R between I, I'	arc appears	$R(I, I')$

Table 1: The four graphical operations in SHAKEN, and their logical equivalent. The examples refer to the axioms listed in the body of this paper.

the user has stated that the invader is the same object as the virus he/she just introduced, by dragging one on top of the other (which then fuse). This asserts *Tangible-Entity2=Virus8*.

Connect: Assert a relation holds between two nodes, by sketching an arc. In Figure 1, the user's action results in $agent(Enter6, Virus8)$ being asserted.

Implications of the User's Assertions

The user's assertions may have logical implications in the KB, and hence may imply changes to the graph the user is viewing. For example, if the user has two graphs for two distinct viruses displayed (similar to the $Virus$ graph in Figure 1), and he/she then unifies the two viruses, this implies (from constraints in the KB) that the two DNA nodes must also be coreferential, and so should also be unified. To feed these changes back to the user, first these "knock on" effects are computed in the KB, and then the graphs the user is viewing are recomputed and redisplayed (preserving as much of the original spatial layout as possible).

Thus SHAKEN is not just a passive graph editor, but is actively engaged in showing the user consequences of his/her assertions when they affect the visible graphs. This is an important and distinctive property of our interface, and necessary to keep the graphs and the KB synchronized so that the dialog remains coherent.

Axiom Synthesis from Graph Operations

Through the above means, the user can only enter ground facts about this particular *example* of his/her new concept. The final stage of this knowledge entry phase (before testing and debugging) is the automatic generalization of those assertions to hold for *all* instances of the user's new concept. This generalization process is algorithmic (rather than inductive), which we now describe.

The axioms which the user has graphically entered are all relationships either between Skolem instances, or between a Skolem instance and a class. For example, the user would enter the earlier assertion **A2** that "the invaded object is the cell" by a 'specialize' graphical operation on the node denoting the invaded object, namely *Tangible-Entity3*. This is illustrated in the first step of Figure 1, and results in the corresponding logical assertion being added to the KB:

$isa(Tangible\text{-}Entity3, Cell)$

To generalize this to apply to *all* instances of the user's new concept $VirusInvasion$, the algorithm behaves as follows:

1. First, the axiom is rephrased to only mention the "root" Skolem instance R, namely the one denoting the concept the user is defining. In our example here, the root Skolem R is *Invade1*, denoting the user's example of *VirusInvasion*. Informally, this means a statement like

   "*Tangible-Entity3* is a Cell"

   is rephrased as

   "the object of *Invade1* is a Cell"

   Note that the latter statement only mentions the root instance $Invade1$. This is required for step 2.

   Formally, each Skolem instance I in the ground assertion is replaced with a variable v, and a formula is added as an antecedent which uniquely identifies v as that Skolem instance I, and no other. In other words, this formula is a *description* of I, stating the unique way it is related to the root R, i.e. is true only when $v = I$. In SHAKEN, this formula is a path (role chain) of relationships from the root instance R to I, found by a simple graph search procedure starting at R and looking for path(s) to I. The resulting formula has the form:

   $p_1(R, x_1) \wedge p_2(x_1, x_2) \wedge ... \wedge p_n(x_{n-1}, v)$

   and thus the rephased axiom has the form:

   $\forall x_1, ..., v\ p_1(R, x_1) \wedge ...p_n(x_{n-1}, v) \rightarrow axiom(v)$

   In the above example, the ground fact

   $isa(Tangible\text{-}Entity3, Cell)$

   would thus be rephased as

   $\forall v\ object(Invade1, v) \rightarrow isa(v, Cell)$

   If there are multiple such objects, then additional predicates are added to the formula until it only holds for I (here *Tangible-Entity3*).

2. This axiom is then generalized so that it holds for *all* examples of the concept $NewC$ being defined. This is done by replacing the root instance R in the axiom with a variable r, and adding an antecedent stating the axiom holds for *all* cases when r is an instance of $NewC$, i.e., when $isa(r, NewC)$ is true. The final axiom will thus have the form:

$$\forall r\ isa(r, NewC) \rightarrow formula(r)$$

where $formula(r)$ is the axiom from step 1 with R replaced by r. In the example, the final result would be:

;;; "The invaded object is the cell."
$\forall r\ isa(r, VirusInvasion) \rightarrow$
$(\forall v\ object(r, v) \rightarrow isa(v, Cell))$

It should be clear that the purpose of step 1, rewriting in terms of R, is to pave the way for step 2, where R is replaced by a universally quantified variable.

One complication must be dealt with for the 'Add' operation. When the user adds a new component to the screen through the 'Add' operation, it is initially disconnected from the root graph describing the user's new concept. This means that there is no path connecting the root instance R to instances in that new graph, and thus the reformulation in step 1 will fail. To handle this, a (graphically invisible) "participant" relation is asserted to hold between R and the new instance whose existence has been declared, stating that the new instance is a "participant" in R. As a result, the procedure in step 1 can now find a path from R to that instance and others in its graph by traversing that participant relation. For example, in graph 3 of Figure 1, the user has added the graph for $Virus$, so the assertion is added to the KB:

$participant(Invade1, Virus8)$

This allows paths to instances in the new graph to be found.

Axiom Synthesis with 'Delete' Operations

An undesirable characteristic of this axiom synthesis routine is that it assumes a monotonically growing KB. As each axiom includes logical descriptions of the objects the user manipulated, generated at a fixed moment in time, the user cannot later delete facts about those objects without risking invalidating those descriptions, and hence his/her earlier synthesized axioms. In the earlier example, if the user were to later delete the assertion *object(Invade1,Tangible-Entity3)*, then the synthesized axiom shown would no longer be valid.

We have recently prototyped (but not deployed) an alternative, and very different, axiom synthesis routine which supports non-monotonic change, thus providing the user with a much desired 'Delete' (of a node or arc) operation, and which we briefly describe here. Rather than converting each user *action* into an axiom, this alternative approach stores the user's *final graph* itself as a (large) "forall...exists..." axiom stating that "forall instances of the concept being defined, all the objects and relationships in the graph exist." This axiom is created only at the end of the user's session, and overwrites any previous axiom for that concept, thus allowing the user to delete as well as add to the graph. To support this, two extensions were needed for the inference engine: First, the user's delete operations must override implications from the KB, to prevent SHAKEN re-inferring the deleted arc/node. Second, when inferencing with several "forall...exists..." statements

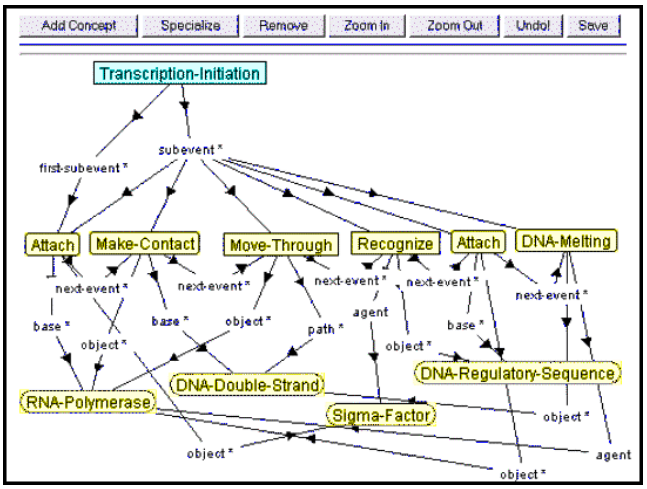


Figure 2: A screendump of SHAKEN's graph interface, showing a user's representation of how the process of RNA transcription is initiated.

like this, the inference engine needs to heuristically determine coreferences between instances in the (logical equivalent of) the multiple graphs, so that they are appropriately merged together. Although these extensions complicate the formal semantics of axioms in the KB, they will provide users with a much-desired 'Delete' capability in later versions of the system. It has become increasingly clear from our experiments that assuming simple axiom semantics and a monotonically growing KB are difficult positions to maintain for real-world knowledge acquisition.

Entering Knowledge through the Interface

Through these operations, the user's task is to assemble a representation of new concepts. The user first provides a name for his/her new concept, then selects from the KB the most appropriate, pre-built generalization of that concept to start from. SHAKEN then displays the initial graph for (an instance of) that new concept. From here, the user adds, specializes, connects, and unifies nodes on the screen to gradually build a representation. An example of one of the simpler representations built by a user during the experimental evaluation is shown in Figure 2. Separate testing and question-answering tools [14, 7] allow the users to debug and pose questions to their representations until they are satisfied.

EVALUATION AND DISCUSSION

During the summer of 2001, an extensive evaluation was performed on SHAKEN, including the graphical entry component described in this paper. Four users who were trained in biology (three graduate students, one undergraduate), and who had no background in programming or formal logic, underwent a week's training in using the system. Following this, they then independently worked over a period of four weeks (except for one who worked for three weeks due to vacation constraints) on encoding an 11-page subsection from a graduate-level textbook on cell biology, including debugging and testing their representations. These trials was run by

an independent contractor (IET, Inc.), rather than ourselves. For the trials, the basic component library was augmented with representations of the prerequisite knowledge needed to understand the subsection. This augmentation was carefully controlled by IET to prevent knowledge from the subsection itself being included in the initial library.

Most significantly, all four users were able to both grasp the basic approach of assembling components, and construct representations using the graphical interface. Over the four week period (three for one user), the users constructed representations of 442 biological concepts (approximately 100 each) ranging in complexity from a single node (i.e., just a concept name) to graphs containing over 100 nodes. The total number of synthesized axioms in the users' final representations (where each axiom-building graphical action results in one axiom being synthesized) were 1408, 567, 1296, and 921 respectively. The users also tested their representations by posing (independently set) questions to them using a menu-driven question-asking interface. The questions were approximately high-school level difficulty, and were mainly "reading comprehension" type questions requiring only simple inference, although a few required more complex inference and simulation. Sometimes this testing revealed errors or inadequacies in the representations, which the users would then correct. The final, system-generated answers to the test questions were collected, and, after the four week period was complete, were scored by an independent biologist on a 0-3 scale (0 = completely incorrect, 1 = mostly incorrect, 2 = mostly correct, 3 = completely correct). At time of writing the final scores are still being tallied, but the evaluators report that the average score is close to 2, reflecting that the users had successfully constructed reasonably accurate, inference-capable representations. These results are significant: they suggest that the basic machinery works, providing a basic vehicle for axiom-building without the users having to encode axioms directly (or even encounter terms like "concept," "relation," "instance," "quantification," etc.); and that those axioms are built in terms of prebuilt knowledge, hence bringing background knowledge into the representations for future reasoning and question-answering tasks. This is an important achievement for this project. In a separate questionnaire to the three users at the end of the four weeks (the fourth user still to complete the questionnaire when back from vacation), all three rated SHAKEN as "useful" as a tool to enter knowledge (on a scale of useless/not so useful/moderate/useful/very useful), and "easy" to use (on a scale of very easy/easy/moderate/difficult/very difficult). This again points to the viability of this approach.

Although the users were able to encode a lot of knowledge with SHAKEN, there was also knowledge they were unable to encode due to the limited expressivity of the interface. The most significant of these, as reported by the users, were:

- simple attribute values (which had to be represented as classes in the current system), e.g., rates, sizes
- equational information e.g., how rates vary with time
- temporal relations, e.g., simultaneous/temporally overlapping events
- pre/post conditions for actions
- richer process models, e.g., repetitive events
- sequences, e.g., nucleotide sequences
- negative information, e.g., being able to say something *doesn't* happen
- locational/spatial information b
- how things change with time (fluent information). The system assumes the graph describes the world at the start of a process, and so, for example, it is not possible to describe what an object looks like at the *end* of a process.

Similarly, comparing the users' source text with what they actually encoded, it is clear that they abstracted away many of the details contained in the text. For example, the source text for the user-built representation in Figure 2 begins:

> "In bacteria, RNA polymerase molecules tend to stick weakly to the bacterial DNA when they make a random collision with it; the polymerase molecule then slides rapidly along the DNA..."

If we compare this text with what was actually encoded (see Figure 2) by one user, we can see that events like "stick" and "slide" have been abstracted to `Make-Contact` and `Move-Through` (whose representations are pre-built in the library), and other phrases like "weakly", "rapidly", "random", and "tend to" have been omitted. (This user has also added an extra prerequisite step, mentioned in text elsewhere, of the sigma factor attaching to the polymerase). In fact, to our surprise, the users seemed to have little or no trouble abstracting out details when building their representations, and they quickly grasped what could be represented and what could not using SHAKEN. In contrast, users (such as ourselves) with more experience in knowledge representation sometimes had more difficulty abstracting in this way when attempting the same encoding task. Interestingly, despite these limitations, the users themselves felt they had managed to encode much of the core knowledge. After the trials were completed, they were each asked: "Next week SHAKEN will be asked questions and answer them using the knowledge you entered, and based on that it will be given a grade. Do you think it will be a passing grade?". All three users reponded quite confidently, saying things like "definitely" and "oh yes". When asked "What kind of grade?", two users answered A-, the third said B. Independent of whether this perception is correct or not, it is interesting that the users themselves felt they had been able to teach the system much of the biological knowledge in the selected subsection.

Another surprise to us was the size of the representations the users created. Some of the users' graphs contained over 100 nodes in, and were rich in relationships and associations. (The users could manage graphs this size as the interface allows them to hide/expose parts of the graph, so not all nodes

need be visible at once). The graph shown in Figure 2 is thus not representative of the typical complexity that the users were able to build. The fact the users were able to build such sophisticated representations perhaps partially explains their confidence in the amount of knowledge encoded.

Despite the reasonable performance scores, there were still errors in the users' final representations. Some of these arose due to the use of linguistic-style devices (e.g., metonymy, analogy, metaphor, approximation) in their graphical assertions. Examples we observed include: indirect reference; interchangeably referring to an object and an event; interchangeably referring to an object and a location; missing coreference statements; overgenerality; missing context (stating a conditional fact as a universal statement); and misuse of case roles. An important future task is to make SHAKEN more active in interpreting and critiquing the users' input, so these errors are detected and corrected more agressively.

A final, interesting point concerns the interaction between representation and question-answering. SHAKEN assumes a single, universal representation for each biological concept, while sometimes the users wanted to be able to represent the same concept in multiple ways, depending on what kind of tasks they wanted their representation to support. Sometimes this resulted in the users creating multiple representations for the same concept (using slightly different concept names). A more principled method for handling different viewpoints like this, either in the KB itself and/or in the reasoning and question-answering procedures, would be desirable.

SUMMARY

We have presented a method for knowledge capture, in which knowledge entry is viewed primarily as a task of component assembly rather than axiom-writing, and shown how it can be implemented using a graph-based interface, based on a novel technique of dialog using examples. Our trials suggest that users can both grasp the approach and construct sophisticated, axiomatic representations, despite having no formal training in logic or AI. This is a potentially significant achievement for enabling subject matter experts to build KBs directly.

ACKNOWLEDGEMENTS

This material is based upon work supported by the Space and Naval Warfare Systems Center - San Diego under Contract No. N66001-00-C-8018. We are grateful to all the other members of the team, working on other parts of the overall system, who have contributed to this work.

REFERENCES

1. K. Barker, B. Porter, and P. Clark. A library of generic concepts for composing knowledge bases. In *Proc. 1st Int Conf on Knowledge Capture (K-Cap'01)*, 2001.

2. J. Blythe, J. Kim, S. Ramachandran, and Y. Gil. An integrated environment for knowledge acquisition. In *Int. Conf. on Intelligent User Interfaces*, 13–20, 2001.

3. A. J. Canas, K. M. Ford, J. D. Novak, P. Hayes, T. Reichherzer, and N. Suri. Using concept maps with technology to enhance cooperative learning in latin america. *Science Teacher*, 2001. (To appear).

4. B. Chandrasekaren. Generic tasks in knowledge-based reasoning: High-level building blocks for expert system design. *IEEE Expert*, pages 23–30, Fall 1986.

5. P. Clark and B. Porter. Building concept representations from reusable components. In *AAAI-97*, pages 369–376, CA, 1997. AAAI.

6. P. Clark and B. Porter. KM – the knowledge machine: Users manual. Technical report, UT Austin, 1999. (http://www.cs.utexas.edu/users/mfkb/km.html).

7. P. Clark, J. Thompson, and B. Porter. A knowledge-based approach to question-answering. In R. Fikes and V. Chaudhri, editors, *Proc. AAAI'99 Fall Symposium on Question-Answering Systems*. AAAI, 1999.

8. DARPA. The rapid knowledge formation project (web site). http://reliant.teknowledge.com/RKF/, 2000.

9. J. Domingue. Tadzebao and webonto: Discussing, browsing, and editing ontologies on the web. In *Proc. KAW'98*, 1998.

10. A. Farquhar, R. Fikes, and J. P. Rice. A Collaborative Tool for Ontology Construction. *International Journal of Human Computer Studies*, 46:707–727, 1997.

11. B. R. Gaines. An interactive visual language for term subsumption languages. In *IJCAI'91*, 1991.

12. B. R. Gaines and M. L. G. Shaw. Webmap: Concept mapping on the web. *The World Wide Web Journal*, 1(1), 1996.

13. W. E. Grosso, H. Eriksson, R. W. Fergerson, J. H. Gennari, S. W. Tu, and M. A. Musen. Knowledge modeling at the millennium. In *Proc. KAW'99*, 1999.

14. J. Kim and Y. Gil. Knowledge analysis of process models. In *IJCAI'01*, 2001. (to appear).

15. S. M. Paley, J. D. Lowrance, and P. D. Karp. A Generic Knowledge Base Browser and Editor. In *Proc. IAAI'97*. AAAI Press, 1997.

16. J. F. Sowa. *Conceptual structures: Information processing in mind and machine*. Addison Wesley, 1984.

17. L. G. Terveen and D. A. Wroblewski. A collaborative interface for browsing and editing large knowledge bases. In *AAAI'90*, pages 491–496, 1990.

18. B. J. Wielinga, A. T. Schreiber, and J. A. Breuker. KADS: A modelling approach to knowledge engineering. *Knowledge Aquisition*, 4(1), 1992.