Graph-Based Acquisition of Expressive Knowledge

Vinay Chaudhri[1], Kenneth Murray[1], John Pacheco[1],
Peter Clark[2], Bruce Porter[3], and Pat Hayes[4]

[1] SRI International, Menlo Park, California, USA
{chaudhri, murray, pacheco}@ai.sri.com
[2] Mathematics and Computing Technology,
Boeing Phantom Works, Seattle, Washington, USA
peter.e.clark@boeing.com
[3] Computer Science, University of Texas at Austin, Austin, Texas, USA
porter@cs.utexas.edu
[4] Institute for Human and Machine Cognition, Pensacola, FL, USA
phayes@ihmc.us

Abstract. Capturing and exploiting knowledge is at the heart of several important problems such as decision making, the semantic web, and intelligent agents. The captured knowledge must be accessible to subject matter experts so that the knowledge can be easily extended, queried, and debugged. In our previous work to meet this objective, we created a knowledge-authoring system based on graphical assembly from components that allowed acquisition of an interestingly broad class of axioms. In this paper, we explore the question: can we expand the axiom classes acquired by building on our existing graphical methods and still retain simplicity so that people with minimal training in knowledge representation can use it? Specifically, we present techniques used to capture ternary relations, classification rules, constraints, and if-then rules.

Categories and Subject Descriptors
H.5.2 User Interfaces – *Graphical user interfaces (GUI)*
I.2.4 Knowledge Representation Formalisms and Methods – *representation languages, predicate logic.*

Keywords authoring tools, knowledge-acquisition tools

1 Introduction

Our goal is to develop tools that enable domain experts to author knowledge bases (KBs) with minimal training in knowledge representation. This goal is important because significant KBs are central to many applications; domain experts lack knowledge-engineering skills, and knowledge engineers lack the domain expertise to replace them.

Previous work on this problem has resulted in a variety of KB editing tools, for example, frame-based editors, such as Protégé[1], OntoEdit[2], Ontosaurus[3], and WebOnto[4]; [5] graphical KB editors, such as the GKB editor[6], the Visual Language (VL) for CLASSIC[7], and task-acquisition methods, such as EXPECT[8].

For the purpose of this paper, we will use the standard knowledge representation terminology of classes, slots, relations, individuals, subclass-of, [10, 11] etc. The work being reported here was done in the context of an object oriented knowledge representation and reasoning system called Knowledge Machine [12].

A common theme in the design of graphical editors such as the GKB-Editor[6] and VL by Gaines[7] has been to construct a graph in which each node represents a class, and labeled edges represent relations or constraints. This approach captures the taxonomic subset of a representation language well, but is limited to just that. In order to capture taxonomic knowledge, rules and their potential interactions, a substantially complex language is required. Using such a language can make it very difficult to work with and understand a knowledge base. Based on our experience in several projects, it is quite apparent that such knowledge bases are of great practical interest for building systems that can answer questions on a wide variety of topics[9]. Effectively addressing the knowledge capture for such knowledge bases requires a shift in the way we view knowledge bases that can be specified in the following two hypotheses:

1. Examples of concepts are a cognitively natural way to capture knowledge.
2. The user's view of the knowledge base should be of a collection of interrelated concepts.

The first hypothesis rules out designs that involve constructing a graph in which each node represents a class, or designs in which a user is presented with an explicitly quantified sentence. The second hypothesis rules out designs that support editing one rule at a time.

Even though the design changes suggested by these two hypotheses are subtle, they are a shift in the way knowledge is presented to a user, and a user's ability to comprehend that knowledge. The approach of viewing a knowledge base as a collection of interrelated concepts is not unique to SHAKEN. Existing graphical editors such as the GKB-Editor and VL also utilize this idea in their design. SHAKEN is, however, unique in using this view as the primary mode of knowledge capture.

In the context of the above two hypotheses, we have been building a knowledge base called the Component Library[13], and a graphical knowledge base editor called SHAKEN[14, 15]. The component library is built by knowledge engineers and contains domain independent classes such as Attach, Penetrate, Physical Object; predefined set of relations such as agent, object, location; and property values to help represent units and scales such as size, color, etc. SHAKEN presents a class to a user as a graph representing an example of the class, and the users construct new classes by connecting instances of the graphs representing existing classes using simple graph manipulation operations such as *add, connect, specialize, etc.*

To illustrate our approach, In Figure 1, we show a graph that captures the concept of Eucaryotic-Cell. Each node in this graph represents an individual, and each edge between two nodes a relationship between them. The class representing each node in the graph is constructed out of more general classes in the component library. For example, Eucaryotic cell is a specialization of Cell, which is constructed from the domain independent concept of Living Entity, which in turn is a Physical Object. In the design of the component library, the knowledge engineers specify pre-built definitions of domain independent concepts such as Physical Object which are inherited by the domain specific concepts shown here. In the component library, the knowledge engineers also specify which slots of a class a user should be usually asked for. This is indicated by a * next to the has-part relation.

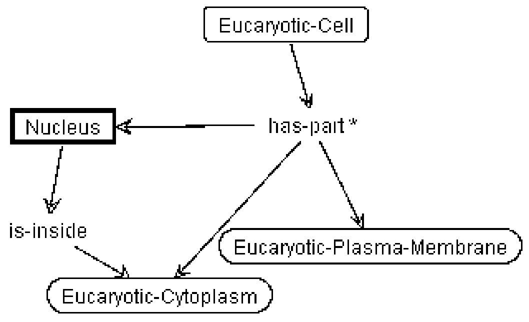


Fig. 1. Graph for the concept Eucaryotic-Cell

This graph is interpreted logically as follows: for every instance of Eucaryotic-Cell, there exists an instance of Nucleus, Eucaryotic-Cytoplasm, and Eucaryotic-Plasma Membrane, such that each is a part of the Eucaryotic-Cell and the Nucleus instance is inside the Eucaryotic Cytoplasm instance[14]. Thus each component description identifies a set of necessary conditions that must hold for each instance of the type of root individual.

If each node in the graph in Figure 1 was represented as a class, we would not have been able to assert the is-inside relationship between Nucleus and Eucaryotic Cytoplasm. This subtle difference is not obvious with a cursory look at Figure 1, and is a key differentiator between SHAKEN and previous graphical knowledge base editors.

We evaluated this system by having subject matter experts (SMEs) capture biology textbook knowledge[16]. These SMEs were first-year graduate students. We trained them with the system for about a week, and asked them to encode a section from a biology textbook. This evaluation revealed that (1) the parsimonious design of this knowledge entry tool was effective in capturing knowledge that supported answering an interestingly broad class of questions, but (2) there were several useful classes of axioms that SMEs wanted to state but the system was unable to handle. For example, consider the following pieces of knowledge.

1. The cell wall is between the nucleus and the virus.
2. If a cell has a nucleus, it is a Eucaryotic cell.
3. If a Christmas tree is taller than 10 feet, and you are in Palo Alto it costs an extra $50.
4. A tank division includes exactly one artillery brigade and at most two armored brigades.

The first example involves capturing ternary relations. The second example captures necessary and sufficient conditions for a class definition, and the fourth example captures constraints on a slot value[17]. The third example involves capturing rules. Taxonomic knowledge, as in the second and fourth examples, could be captured in prior tools such as VL[7]. Our approach differs, however, in its emphasis on expressing knowledge in terms of instances. We capture the same knowledge without forcing the user to use abstractions, in a context where examples are used to convey concepts and express relationships. This makes the knowledge capture cognitively simpler than the approach considered in Visual Language.

The evaluation results presented a challenge: How can we expand the basic design to support these classes of axioms while preserving the system's simplicity and ease of use? We next present our solutions, justify them, and present an evaluation of their use by SMEs.

2 N-Ary Relations

Traditional semantic network representations, and consequently the graphical editors supporting them, have been dominated primarily by binary relationships. Therefore, capturing knowledge that references relations of arity higher than two is a new challenge. In Figure 2, we illustrate how ternary relations are captured in our graphical description of an example of a class; in this example, the cell wall is in between the nucleus of the cell and the attacking virus. The *is-between* hyper-edge represents an atomic statement with a ternary predicate: as usual, the incoming arc indicates the first argument; the two outgoing arcs, from left to right, are the second and third arguments. The hyper edges are identified separately by showing them in bold italics.

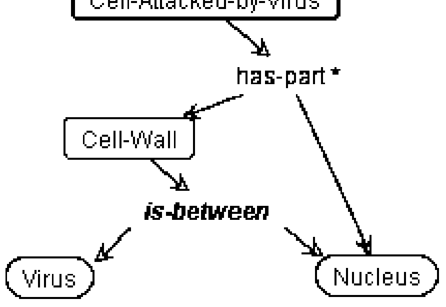


Fig. 2. Graphical presentation of a ternary relation

The *has-part* relation participates in two edges and represents two atomic statements that share a common first argument, indicated by the incoming arc. Each of the two outgoing arcs represents the second argument of the statement. Thus our approach to handling high-arity relations involves adopting a second display convention for representing statements with graph edges: the second argument is indicated by an outgoing arc from the bottom-left corner and the *nth* argument is denoted by an arc going from the bottom-right corner, with the remaining arguments indicated by the arcs emanating and spread equally between the bottom-left and the bottom-right corners. This approach gracefully expands the expressiveness of the basic graph formalism that traditionally uses binary edges.

3 Sufficient Conditions

We now consider axioms that define the sufficient conditions for an individual to be a member of a class. An example of such an axiom is *If a cell has a nucleus, it is a Eucaryotic cell.* We refer to such axioms as classification rules. SHAKEN enables a SME to capture a classification rule for a class *A* by selecting a subset of *A's* necessary conditions as being sufficient to classify any instance of the direct super classes $B_1, \ldots, B_n$ of *A* as also being an instance of *A*.

While defining a class A, the first step is to define its super classes $B_1, \ldots, B_n$. This is accomplished using a form-based dialog through which the user gives a name to the class A, and specifies its super classes. We assume that the user has specified the super classes, and now they are ready to define the necessary and sufficient slot values.

The process for defining sufficient conditions requires a simple and intuitive extension to our existing interface[8]. First, the user selects a subset of the nodes in the graph to be treated as a group; the interface responds by displaying a rectangle that includes all selected nodes. Next, the user designates that the group of nodes defines a sufficient condition for that class, and the system synthesizes a classification axiom from the grouped nodes and edges.

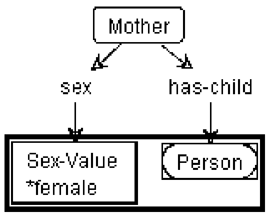


Fig. 3. Example class with sufficient condition

Figure 3 presents the graph describing an example of class Mother. This description also includes a sufficient condition indicated by the green rectangle around the nodes representing the individuals Sex-Value and Person. Since Figure 3 is an example of class Mother, the node Person is in fact an individual and should not be confused with a type constraint on the value of the slot has-child. Figure 4 shows a KIF[11] representation for the rule captured by the example description of this graph.

```
(implies
    (isa ?person Mother)
        (exists (?child ?sex-value)
            (and (isa ?child Person)
                (isa ?sex-value Sex-Value)
                (sex ?person ?sex-value)
                (value ?sex-value *female)
                (has-child ?person ?child))))
(implies
    (and (isa ?person Person)
        (sex ?person ?sex-value)
            (value ?sex-value *female)
            (has-child ?person ?child))
        (isa ?person Mother))
```

Fig. 4. KIF for the class Mother

An alternative approach for capturing sufficient conditions could have been to introduce a new view on the example graph that exposes only sufficient properties. With such a design, there would have been no need to identify a subset of nodes that represent sufficient properties. We did not consider that approach in our initial design since one of our goals is that all knowledge capture should occur in the context of an example of a class.

4 Capturing rules

Consider rules such as

If a terrain has a width that is less than 10 feet, then it has restricted trafficability for tanks.

If a Christmas tree is taller than 10 feet, and you are in Palo Alto it brings a premium of $50.

If it is raining or snowing, the person going out will wear a coat.

The kinds of rules shown above occur quite frequently. Sometimes, it is possible to force them into class definitions by defining more specific classes, such as a class representing Restricted Trafficability Terrain and Tall Christmas Trees, but very often, such classes are not natural, or too many, and a more direct encoding of rules is desired.

We capture this knowledge by introducing a *conditional edge*. For example, to specify the clothes worn, we simply state its value as a *Coat*, but we make this value conditional. The knowledge is stated in the context of an example instance of Go-out. We illustrate this in Figure 5. The diamond indicates that the value of an edge is conditional. In the implemented system, the user can roll over the diamond and get a description of the condition. In the current version of the system, the conditions may be associated with only one edge. If a condition applies to multiple edges, it must be stated separately for each edge.

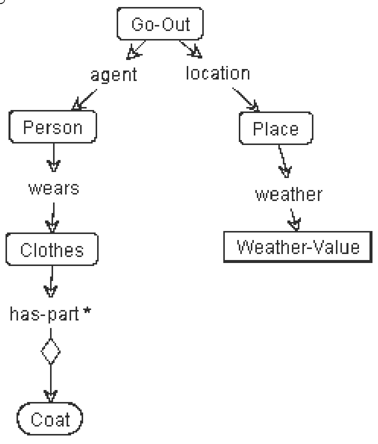
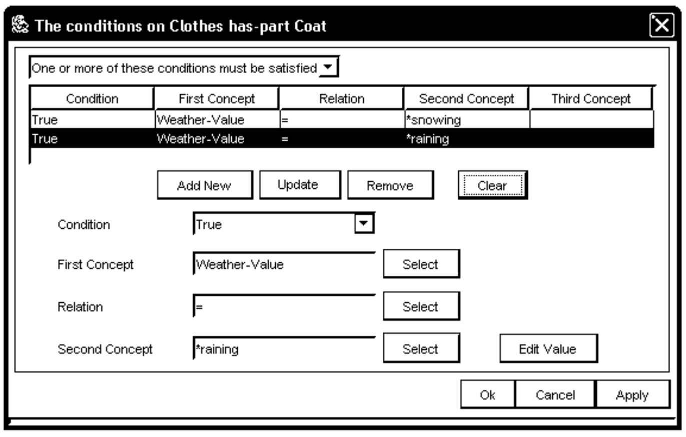


Fig. 5. Conditional rule that illustrates that if it is snowing, while going out, one should wear a coat



The conditions on Clothes has-part Coat

One or more of these conditions must be satisfied

Condition	First Concept	Relation	Second Concept	Third Concept
True	Weather-Value	=	*snowing	
True	Weather-Value	=	*raining	

Add New · Update · Remove · Clear

Condition: True

First Concept: Weather-Value · Select

Relation: = · Select

Second Concept: *raining · Select · Edit Value

Ok · Cancel · Apply

Fig. 6. Example slot-value condition

Figure 6 illustrates the authoring of a condition on an edge. The user will invoke this dialog when they need to author a piece of knowledge involving conditional knowledge. This relies on the assumption that the user has been trained to identify which tool is to be invoked when.

```
(implies
   (isa ?action Go-Out)
      (exists (?person ?place ?weather ?clothes)
         (and (agent ?action ?person)
            (location ?action ?place)
              (wears ?person ?clothes)
              (weather ?place ?weather)
              (value ?weather ?weather-value)
                 (implies
                    (or
                       (equal ?weather-value *raining)
                       (equal ?weather-value *snowing))
                    (exists (?coat)
                       (has-part ?clothes ?coat))))))
```

Fig. 7. KIF[11] representation of the graph shown in Figure 5 and the associated conditional rule as entered in Figure 6

The dialogue of Figure 6 uses tables instead of a graphical method for capturing conditions. There is no deep reason for this other than the observation that it was straightforward to design a table-based interface in which a condition could be specified by pointing to the nodes in the graph, and that multiple conditions could be entered and edited. Therefore, for a value to be filled in the table, it must be a part of the graph. This works in practice because the graph representing an example of a class includes all related individuals that might participate in a condition. The table-based interface, by itself, is not novel, and is found in other knowledge base editors such as Protégé and GKB-Editor. Its use in conjunction with the example description of a class is novel in that when the user fills in various values in the table, they choose those values by pointing to the nodes in the graph.

Our support for capturing slot-value conditions is sufficiently expressive to author a large and useful subset of the SHAKEN Action Description Language (SADL), [18] however, to preserve simplicity we do impose restrictions. The conditions cannot be nested. For example, it is not possible to state an axiom that contains a conjunction and a disjunction: *If a Christmas tree is taller than 10 feet, and you are in Palo Alto or San Francisco, it costs an extra $50.* SADL is comparable to many other standard process languages, and is seamlessly integrated into our KM system. Its role is transparent to the users whenever they are authoring process knowledge.

5 Constraints

We consider only three types of constraints: type, numeric range, and cardinality constraints that apply to just one slot. We do not handle constraints that apply to multiple slots, or to several classes. The graph of Figure 8 specifies that a tank division includes an artillery brigade and two armored brigades and two Mechanized Infantry

Brigades; it does not indicate that there are no more than one artillery brigade and two armored brigades. It is often useful to denote what cannot be true in a domain by restricting the values of slots with constraints. In the current interface, we support two kinds of slot-value constraints: (1) *element constraints* apply to each distinct value of the slot, and (2) *set constraints* apply collectively to the set of known values for a slot. The element constraints we currently support include type constraints *must-be-a* and *mustnt-be-a*, and the set constraints we support include *exactly*, *at-least*, and *at-most*; see [12]for the semantics of these constraints.

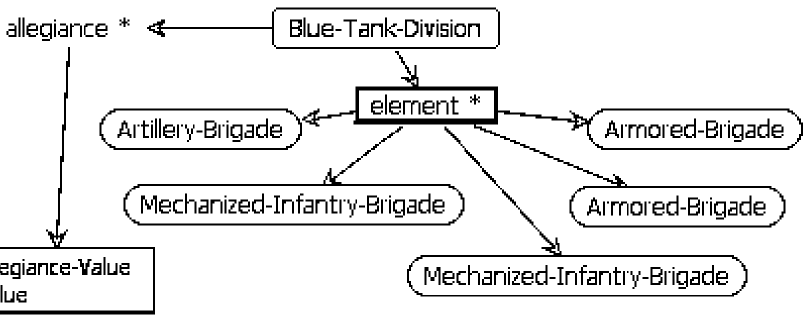


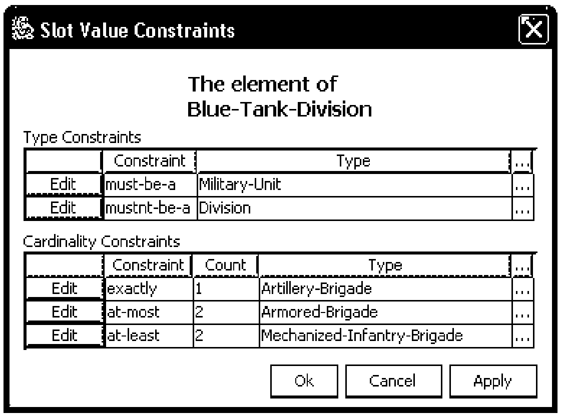
Fig. 8. Graph for Blue-Tank-Division

Figure 9 illustrates an example of the constraints that might appear on the destination (i.e., the second argument) of the *elements* edge for the class Blue-Tank-Division; we have adopted a tabular format for capturing constraints. The type of constraint (e.g., *must-be-a, mustnt-be-a, exactly*) is selected from a pull-down menu. The numeric values for constraints can be simply typed in the dialog box. The user can visualize the constraints that have been entered by right clicking on a slot label that has an option to display edge constraints.

Just like the condition editor, a SME would bring up this dialog when they need to state type or cardinality constraints about a class. This assumes that they have been trained to identify the constraints, and to invoke the constraint editor at an appropriate time.

An alternative way to model constraints would have been to display them on the same graph on which we are displaying the example of a class. We chose not to use that approach because the constraint information is conceptually a higher order information about the example, and we chose to not mix that with knowledge about the example itself.

The dialog of Figure 9 is fairly primitive in that it supports type, cardinality, and numeric constraints only. We implemented this set because our knowledge representation system [12] supports only those constraints. The basic design of Figure 9 can be generalized to a richer set of constraints, and in fact, many existing systems such as Protégé and the GKB-Editor support more elaborate interfaces for editing constraint information.


Slot Value Constraints

The element of
Blue-Tank-Division

Type Constraints

	Constraint	Type	...
Edit	must-be-a	Military-Unit	...
Edit	mustnt-be-a	Division	...

Cardinality Constraints

	Constraint	Count	Type	...
Edit	exactly	1	Artillery-Brigade	...
Edit	at-most	2	Armored-Brigade	...
Edit	at-least	2	Mechanized-Infantry-Brigade	...

Fig. 9. Capturing Slot-Value Constraints

Figure 8 also illustrates another way a graph that represents an example of a class might differ from a graph that represents the class itself: A Blue-Tank-Division contains two instances of Armored-Brigade which are explicitly enumerated. Underneath these two nodes are two unique skolem individuals each representing a different Armored Brigade. We do not expose that level of detail to the users. The users can choose to distinguish between the two Armored-Brigades by assigning them a new label that is meaningful to them.

6 Evaluation

We have tested the extensions described here in two different application domains with different sets of users. Our claim in reporting the results of these studies here is that they give evidence that the capabilities proposed in the paper enable SMEs to capture knowledge. We do not claim that these are the best knowledge-capture capabilities possible, or that the results discussed next are a conclusive proof that these extensions will work with any set of users. We do not compare the version of the system with the extensions presented in the paper with the version that did not have those extensions, because the extensions represent new functionality which would not have been otherwise possible.

We tested the use of sufficient conditions during the fall of 2002 in the context of a military course of action (COA) analysis problem[19, 20]. We tested the slot value conditions, and constraint editing capabilities with domain exerts in physics, chemistry, and biology during January of 2004. We summarize here the highlights of these evaluations to substantiate the claim of this paper: we have effectively expanded the class of axioms captured by the interface and still retained its simplicity.

A military commander uses a COA to communicate to her subordinates one way to accomplish a mission. They then evaluate multiple competing COAs using appropriate comparison criteria and decide on one to build into a complete action plan for the mission. In the test the knowledge-capture task was to express the knowledge necessary to critique the COAs.

We retained two U.S. Army officers to conduct the evaluation. They used the necessary and sufficient conditions to specify the required combat power in given situations. During the evaluation, both SMEs quickly became adept at using the SHAKEN interface for capturing knowledge in the forms of class descriptions, and necessary and sufficient conditions. They collectively authored 56 additions to the critiquing knowledge (e.g., new classes or rules), including 13 new classes. An example of necessary and sufficient condition authored by the SMEs is suggested in the following piece of knowledge: When an aviation unit attacks an artillery unit, it is enough to have a combat power ratio of 0.3; SMEs captured this knowledge by defining the example of a class in which an aviation unit attacks an artillery unit. The sufficient conditions for such a class are that the agent is an aviation unit and the object is an artillery unit. Whenever an action is encountered that meets these sufficient conditions, it automatically gets classified as an instance of this class, and assigned a combat power ratio of 0.3. This example is illustrated in Figure 10 below.

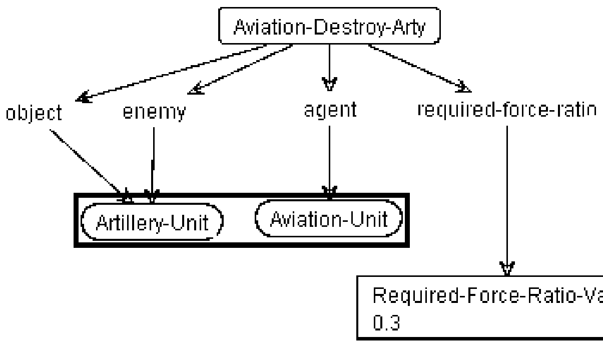


Fig. 10. Example of Classification Rule for an Aviation-Unit Attacking an Artillery-Unit

The evaluation showed that the SMEs successfully authored both new classes and new classification rules; however, there were a few problems observed in their attempts. We consider here two such examples. In one, the user meant to include two nodes while constructing a classification rule but selected, and thus included, only one node. In a second example, the user included four edges that failed to capture the intended classification rule; yet the desired rule could have been captured simply by including two nodes. This reflects a deficiency in the training that did not make it sufficiently clear that defining sufficient conditions requires including the relevant nodes. While many of the SME-authored sufficient conditions effectively captured the intended rule and supported useful inference, it is apparent that sometimes the SMEs did not clearly understand what classification rule resulted from the nodes or edges they selected.

In January 2004, we conducted a 1-day experiment with subject matter experts in physics, chemistry, and biology. The knowledge-authoring teams consisted of a domain expert, a usability expert, and a knowledge engineer. The domain expert was the knowledge author, the usability expert observed the kind of difficulties encountered by the SME, and the knowledge engineer answered questions and requests for assistance during the knowledge entry process. The first half of the experiment involved an intensive step-by-step training session, followed by an open-ended knowledge construction exercise.

In the open-ended exercise, the biology SME focused her knowledge authoring around the concept of Bacterial-Protein-Translation. Using a biology textbook to guide her knowledge entry, she constructed the graph shown in Figure 11. Central to the concept of protein translation is an iterative process where amino acids are linked together to form a polypeptide. In capturing this, the SME opted to write conditions to identify when the cycle of events should continue and when it should be terminated. In the figure below one can see that after the sub-event of "Recognition" there is a cycle of events: "Come-Together," "Catalyze," and "Slide." There are conditions, marked with diamonds, on the next-event edge between "Slide" and "Come-Together" and between "Slide" and "Divide." The condition, which is not shown in this view, states that if the destination of the slide event is a "Protein-Stop-Codon," then the next-event is "Divide," otherwise the next-event is a "Come-Together." This SME, with little to no prior training in logic and programming, with only a few hours of training on the system and a moderate, though certainly not insignificant, amount of help from the knowledge engineer, was able to capture expressive knowledge that pushed the limits of the system. This success with complicated concepts and advanced tools is evidence of the potential of the system to empower domain experts with the ability to effectively communicate their knowledge to the system.

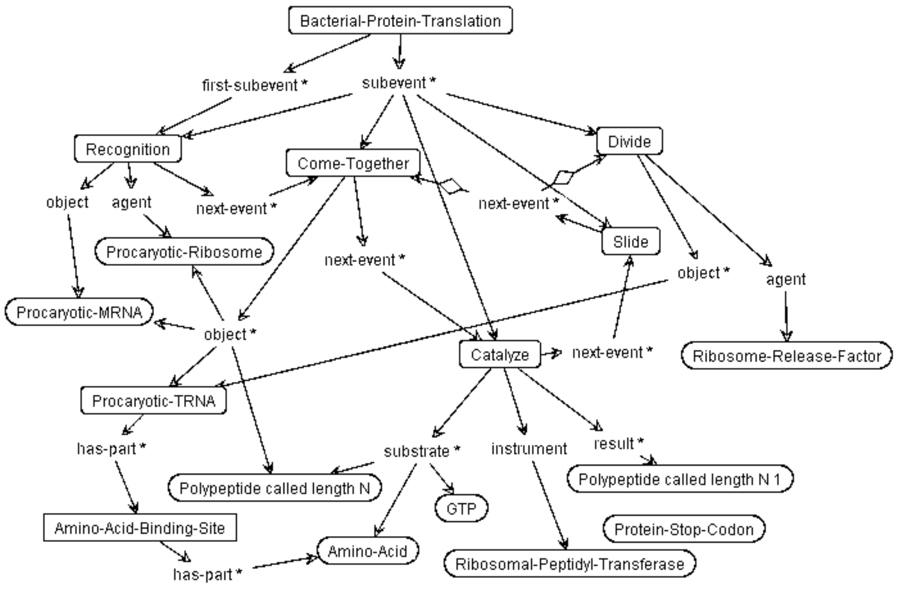


Fig. 11. Concept of Bacterial-Protein-Translation Authored by SME at January Experiment

The SME who created the above graph in the experiment had this to say about the experience, “When I first encountered the SHAKEN system, I anticipated that I would need a great deal of technical and programming knowledge in order to enter content information into the system. I was pleased to discover that once I learned a few rules and procedures, I was able to construct a reasonably detailed map of a multistep process. I did need some technical help navigating some of the more complex steps, but I think the end result was a workable map. It was very useful to have a library of specific terms already available in my content area, so I was able to assemble my map without having to define each noun and verb individually.

We have clearly expanded the class of axioms that can be captured using the system. An important and significant class of knowledge that remains to be captured involves following a detailed computational algorithm, for example, the procedure to compute the pH of a solution. Examples of knowledge that our designs are focusing on are equations and knowledge about how and when to apply them, policies, and a very limited set of visual programming tools trying to capture notions like iteration and decision making.

There are several other aspects of the knowledge capture problem that we have not explicitly considered in this paper. For example, the editing of taxonomies is supported by SHAKEN using a form based interface that is not particularly unique. SHAKEN does not, however, allow domain experts to create or edit relations. In the current system, this responsibility resides with the knowledge engineers. Before an SME starts to define concept descriptions, there are early phases of design in which the SME identifies relevant concepts, and breaks down the knowledge to be formalized into smaller chunks to be formalized [21]. The current SHAKEN system does not address the early phases of the knowledge capture process. As the knowledge capture is in progress, the user needs to be proactively supported, perhaps, by a mixed initiative dialog [22].

A possible question that arises based on the current experience is: whether graphs are a compelling way to capture knowledge? Even though there is success in using graphs for knowledge capture, the graphs as considered here are not a natural form that the knowledge occurs in the real world. The real world knowledge is found in form of text, diagrams, pictures, tables, formulas, etc. Given that we will not have an automatic translation from text to logic any time in the near future, the graphs offer a middle ground. The current work suggests that the graphs are a useful middle ground, but at the same time, we believe that they need to be complemented where more compelling visualizations such as tables or equations may apply. That is one of the topics for future extensions to SHAKEN.

7 Related Work

We will now compare our work with an earlier work on the VL system, since this system bears the most resemblance to SHAKEN and provides the most relevant framework for comparison. Both SHAKEN and VL [7] are motivated to support use by "non-programmers" (e.g., SMEs) and yet be formal and (unlike many of the early implementations of semantic networks as KRLs) precise.

The primary difference between VL and SHAKEN is that the knowledge entry in SHAKEN is driven by examples: that is, knowledge is captured by defining instances in an example of the concept being defined. This allows SHAKEN to capture more complex forms of axioms such as in Figure 1, 3, and 5 that cannot be captured using VL. Figure 1 shows the closest that VL can come to representing the knowledge of Figure 1, however, the fact that the nucleus is inside the cytoplasm cannot be expressed since VL uses classes and not instances.

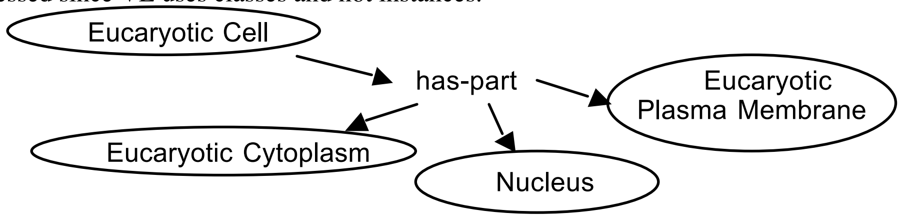


Fig. 12. Eucaryotic Cell as written in VL

Since SHAKEN graphs represent examples of classes, each node corresponds to an individual in the KB depicting an entity or event or property value, and each arc is directed and corresponds to a slot in the KB. In contrast, the VL includes several types of nodes (e.g., constraints and rules) that do not naturally correspond to the entities and events and property values in the domain being represented, as well as both directed and undirected arcs. Furthermore, arcs are unlabeled and the back-end denotation of each arc is substantially determined by the types of the nodes connected by the arc and whether or not the arc is directed. For example, in the graph from VL, Figure 13,

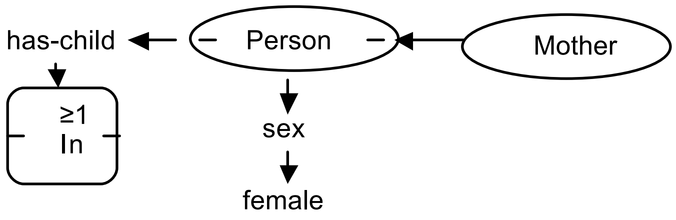


Fig. 13. Concept Mother as written in VL

represents the necessary and sufficient conditions for the class Mother that we considered earlier in Figure 3. The nodes in this graph, Mother, and Person are classes. The plain oval represents a defined class (e.g., Mother), and an oval with dashes on the side represents a primitive class (e.g., Person). The labels on the edges are relations. The values in the rounded box represent a constraint that the person has at least one child. The value of the slot sex is female. The advantage in the VL visualization is that it makes it explicit that Mother is a subclass of Person. The SHAKEN visualization leaves this information implicit because we do not explicitly represent classes in the graph. The fact that the super class of Mother is a Person is available when a user rolls the mouse over the Mother node. An apparent disadvantage of the VL representation that is based on our interpretation of the cited reference [7] is that if a defined class has some properties in addition to the defining properties, VL does not give a way to distinguish between the two. SHAKEN accomplishes this by providing the group node notation.

It is not possible to express in VL ternary relations or the complex rules considered in Figures 2 and 5. VL allows only rules that relate two classes. For example, consider another graph from VL, shown in Figure 14.

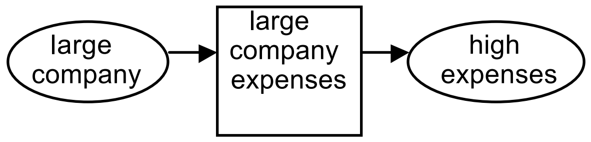


Fig. 14. Large Company Expenses rule as written in VL

This graph represents a rule named Large Company Expenses that applies to a Large Company, and asserts that its expenses are high. The same rule is expressed in SHAKEN as shown in Figure 15: We assert using the condition editor a conditional rule stating that if the size is large, the expenses must be high.

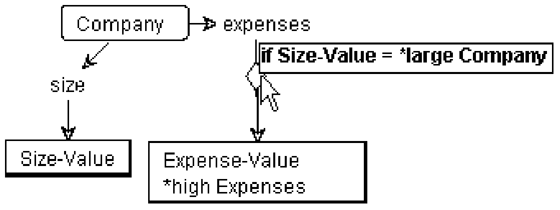


Fig. 15. Large Company Expenses rule as written in SHAKEN

The VL system uses a composition operation that allows the construction of large knowledge structures from smaller ones. The composition is achieved by grouping nodes with the same label together and defining a concept in terms of the outgoing arrows from the nodes. SHAKEN and VL approach are similar in that they both allow the visualization of knowledge as a collection of interrelated concepts.

Another distinctive aspect of our approach is that it is strongly dependent on the prior knowledge encoded in the library. The prior knowledge restricts the existing concepts that could be connected together, and also the relations with which they could be connected. Thus, unlike other graphical editors, the users do not draw arbitrary graphs: they have to use something that is already in the library.

We would like to compare the SHAKEN system to an alternative like Protégé[1]. Protégé is not designed with the intention of being primarily used by SMEs, and its interface can be customized to different domains.

8 Summary

We have motivated and presented several knowledge-authoring extensions to SHAKEN. These extensions support capturing a substantially greater variety of axiom classes than was previously possible. Hyper-edges enable capturing ternary (and higher) relations. Sufficient conditions as well as necessary conditions can be expressed in the graph structure. Slot-value conditions enable capturing important classes of problem-solving rules. Constraints limit the assertions that may be entered in the knowledge base. These pieces, defined functionally for the user, can be combined with one another to create robust and detailed knowledge. The extensions use an interface designed to increase the expressiveness of the acquirable knowledge while remaining faithful to our mandate of providing tools that support knowledge authoring by SMEs.

Acknowledgments

This research was supported by DARPA's Rapid Knowledge Formation project[23] under contract N66001-00-C-8018. We are grateful to Peter Yeh, Jim Blythe, Jihie Kim, Tomas Uribe, Yolanda Gil, Michael Eriksen, Chris DiGiano and the usability experts, and the many other members of our collaboration team.

References

1. Gennari, J., et al., The Evolution of Protege: An Environment for Knowledge-Based Systems Development. International Journal of Human-Computer Interaction, 2003. 58(1): p. 89-123.
2. Sure, Y., S. Staab, and J. Angele. OntoEdit: Guiding ontology development by methodology and inferencing. in International Conference on Ontologies, Databases and Applications of Semantics ODBASE 2002. 2002. University of California, Irvine, USA: Springer, LNCS.
3. Sowa, J., Knowledge Representation: Logical, Philophical, and Computational Foundations. 2000, Pacific Grove, CA: Brooks Cole Publishing Co.
4. Domingue, J., Tadzebao and WebOnto: Discussing, Browsing, and Editing Ontologies on the Web, in Proc. KAW'98. 1998
5. Aaronson, L., et al., Artifactory Research Resources. 2004, Artifactory research group at SRI http://www.ai.sri.com/artifactory/resources.html#Editors.
6. Paley, S. and P. Karp, GKB Editor User Manual. 1996.
7. Gaines, B.R., An Interactive Visual Language for Term Subsumption Languages, in IJCAI'91. 1991
8. Blythe, J., et al., An Integrated Environment for Knowledge Acquisition, in Int. Conf. on Intelligent User Interfaces. 2001. p. 13--20
9. Barker, K., et al., Halo Pilot Project. 2003, SRI International: Menlo Park. p. 35, http://www.ai.sri.com/pubs/full.php?id=990.

10. Chaudhri, V.K., et al., OKBC: A Programmatic Foundation for Knowledge Base Interoperability, in Proceedings of the AAAI-98. 1998: Madison, WI
11. Genesereth, M.R. and R.E. Fikes, Knowledge Interchange Format: Version 3.0 Reference Manual. Jun 1992(Logic-92-1).
12. Clark, P. and B. Porter, KM -- The Knowledge Machine: Reference Manual. 1999.
13. Barker, K., 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. p. 14--21
14. Clark, P., et al., Knowledge Entry as the Graphical Assembly of Components, in Proc 1st Int Conf on Knowledge Capture (K-Cap'01). 2001. p. 22-29
15. Thomere, J., et al., A Web-based Ontology Browsing and Editing System, in Innovative Applications of Artificial Intelligence Conference. 2002. p. 927--934
16. Schrag, R., et al. Experimental Evaluation of Subject Matter Expert-oriented Knowledge Base Authoring Tools. in 2002 PerMIS Workshop. 2002. Gaithersburg, Maryland: National Institute of Standards and Technology.
17. Brachman, R. and J.G. Schmolze, An overview of the KL-ONE knowledge representation system. Cognitive Science, 1985. 9(2): p. 171-216.
18. Blythe J., e.a., SADL: Shaken Action Description Language. 2001 http://www.isi.edu/expect/rkf/sadl.html.
19. Barker, K., et al., A Knowledge Acquisition Tool for Course of Action Analysis, in Innovative Applications of Artificial Intelligence Conference. 2003. p. 34--50
20. Pool, M., J.F. K. Murray, M. Mehrotra, R. Schrag, J. Blythe, and H.C. J. Kim, P. Miraglia, T. Russ, and D. Schneider. Evaluation of Expert Knowledge Elicited for Critiquing Military Courses of Action. in Proceedings of the Second International Conference on Knowledge Capture (KCAP-03). 2003.
21. Boicu, M., et al. Mixed-initiative Control for Teaching and Learning in Disciple. in IJCAI-03 Workshop on Mixed-Initiative Intelligent Systems. 2003. Acapulco, Mexico.
22. Kim, J. and Y. Gil. Proactive Acquisition from Tutoring and Learning Principles. in AI in Education. 2003.
23. DARPA, The Rapid Knowledge Formation Project. 2000 http://reliant.teknowledge.com/RKF/.