Visually Interacting with a Knowledge Base Using Frames, Logic, and Propositional Graphs

Daniel R. Schlegel and Stuart C. Shapiro

Department of Computer Science and Engineering
and Center for Cognitive Science
and Center for Multisource Information Fusion
University at Buffalo, The State University of New York, Buffalo NY 14260, USA
{drschleg,shapiro}@buffalo.edu

Abstract. The knowledge base of a knowledge representation and reasoning system can simultaneously be thought of as being logic-, frame-, and graph-based. We present a method for naturally extending this three-fold view to methods for visual interaction with the knowledge base in the context of SNePS 3 and its newly developed user interface. Addition to, and querying of, the knowledge base are tasks well suited to a frame or logical representation. Visualization and exploration on the other hand are best done through the use of propositional graphs. We show how these interaction techniques, which are extensions of the underlying knowledge base representation, augment each other to allow users to manipulate and view large knowledge bases.

1 Introduction

The knowledge base (KB) of a knowledge representation and reasoning (KR) system can be conceived of simultaneously as a set of logical expressions, as a set of frames, and as a graph. This allows the user interface for the system to present and allow interaction with the KB in any of these three different ways. The KB of the SNePS 3 KR system [17] is no exception, and the recently developed SNePS 3 user interface makes use of all three paradigms to provide the user with what we think are the most helpful ways to graphically interact with the KB. In this paper, we focus on the presentation of a SNePS 3 KB as a graph, and on the way that conceiving of it as a set of frames and logical expressions influences the visualization of the graph.

The SNePS 3 graphical user interface (GUI), shown in Fig. 1, highlights the visualization of a SNePS 3 KB as a propositional graph. It allows the user to add to the KB using either the logic or frame paradigm, and to query and select a part of the KB for graphical display using frames in a relational database-esque visual query by example (QBE) way. The user can also explore the KB by expanding or hiding specific parts of the graph that are currently visible. Binary relations in the graph can be collapsed to provide a less cluttered display without changing the semantics of the graph. In this way, most of the interaction with the knowledge base in terms of asserting and querying is carried out in a frame-based way, but visualization of the result is largely graph-based.

In Sect. 2 we introduce SNePS 3, its three-fold knowledge base representation, and the relationships between those three views. We discuss frame-based and graph-based

M. Croitoru et al. (Eds.): GKR 2011, LNAI 7205, pp. 188–207, 2012.

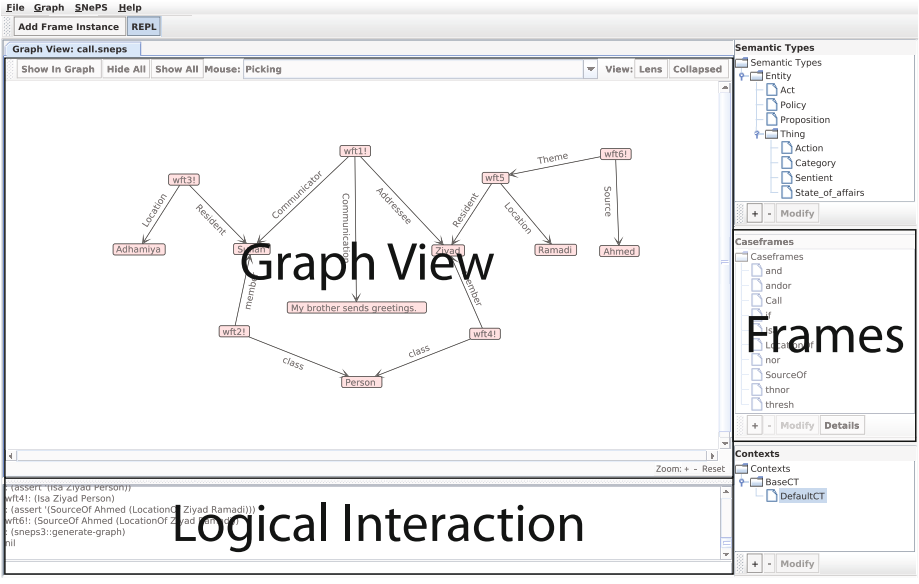

Fig. 1. The SNePS 3 GUI supports graph-, logic-, and frame-based interactions

interactions with the knowledge base in Sects. 3 and 4, respectively. Section 5 discusses the issues surrounding large KBs, and Sect. 6 introduces the concept of collapsing binary relations in the graph. Finally the usefulness of the system will be considered along with a comparison with ontology editing software in Sect. 7.

SNePS 3 is implemented in Common Lisp. The GUI is implemented in Java. The Lisp-Java interface is implemented using jLinker [5].

2 SNePS 3

2.1 Logical Expressions

A SNePS 3 KB may be thought of as a set of logical expressions. For example, the first eight lines of Fig. 2 show a user using the logical interaction pane of the SNePS 3 GUI to assert four atomic propositions, and the GUI echoing them back, each labeled with an internal name of the form `wft`i`!`. The choice of "`wft`" to begin the name will be explained shortly. The "`!`" is appended to the name to indicate that the proposition is asserted (considered to be True in the KB). The syntax used for logical expressions is a version of CLIF [8], and includes: `not`; `if`; the set-oriented connectives `and`, `or`, `nand`, `nor`, `xor`, `iff`, `andor`, and `thresh` [19]; and arbitrary and indefinite terms [18].[1]

To facilitate metaknowledge—knowledge about knowledge [20], propositions are considered to be first-class objects in the semantic domain, and all well-formed SNePS

[1] SNePS 3 examples shown in this paper will be limited to ground atomic propositions.

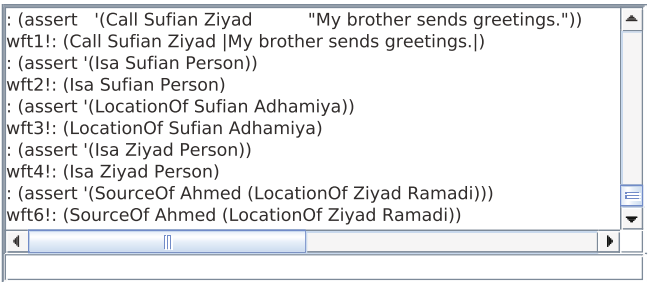

Fig. 2. A set of assertions in the text-based logical interface to SNePS 3

logical expressions, including those that look like formulas, are terms [16]. So what look like predicates (for example, "`Call`", "`Isa`", "`LocationOf`", and "`SourceOf`" in Fig. 2) are actually proposition-valued functions. Even "logical connectives" are function symbols. The internal names of SNePS 3 expressions start with "`wft`" as a reminder that they are "well-formed terms." This is illustrated in the last assertion of Fig. 2, which is intended to represent the proposition that Ahmed is the source of the proposition that the location of Ziyad is Ramadi. Taken together, the assertions of Fig. 2 are intended to mean that *"Sufian, a person in Adhamiya, called Ziyad, a person who, according to Ahmed, is in Ramadi, saying 'My brother sends greetings.' "*

2.2 Frames

Before a function symbol can be used, it must be associated with a caseframe. A caseframe has a *name*, specifies a *semantic type* and *slot* for each argument of the function, and is associated with one or more function symbols. If the caseframe is associated with a single function symbol, that function symbol is used as the name of the caseframe. The case of a caseframe associated with multiple function symbols is discussed below. The function symbol `Isa` is predefined in SNePS 3 as associated with a caseframe whose slots are `member` and `class` and whose semantic type is `Proposition`. The slots in a caseframe are ordered to correspond with the order of the arguments of the function it is meant to represent.

The user interface allows for the definition of such a caseframe. Figure 3, for example, shows the `Call` caseframe being defined. The user selects the slots they are interested in and moves them to the right column. The order they are in from top to bottom represents the order of the slots in the caseframe. This defnition is automatically converted to its appropriate representation in the logical interface to SNePS 3 and displayed in the user interface (see Fig. 4). A port of SNePS 3 to the Clojure [7] programming language is in progress, and will allow the user interface to interact with the underlying data structures of SNePS without this extra translation step. Also displayed in the user interface is the current listing of defined caseframes, which in Fig. 5 now includes the `Call` caseframe.

Additional slots may be defined by the user. A slot definition includes a name, the minimum and maximum number of terms that may fill the slot, and the semantic type

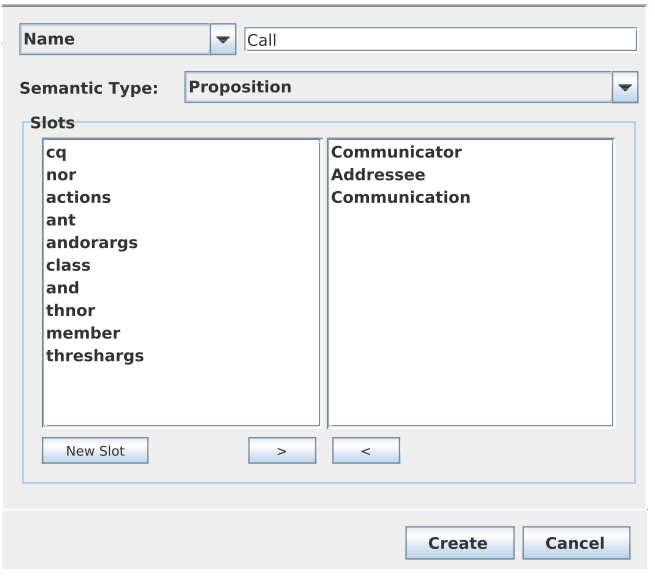

Fig. 3. The user interface for defining the `Call` caseframe of type `Proposition`, populated with the slots `Communicator`, `Addressee` and `Communication`

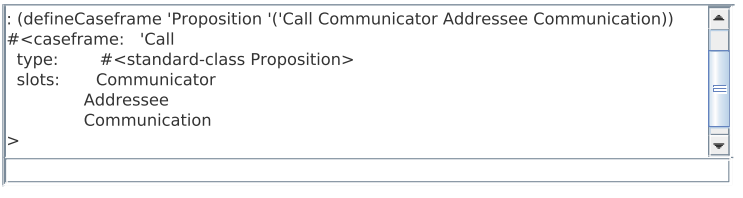

Fig. 4. The logical interface displaying the execution of the input in Fig. 3 in the logical language

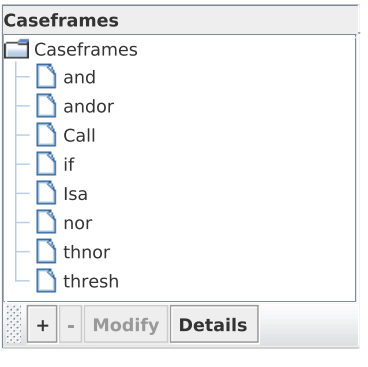

Fig. 5. The list of defined caseframes, including the `Call` caseframe

of the fillers. However, explaining the details of slot definitions would go beyond the scope of this paper.

A *frame* is an instance of a caseframe. That is, the term (F x_1 ... x_n) is represented by an instance of the caseframe whose name is F, whose semantic type is the type specified when defining F, and whose slots, $s_1, ..., s_n$ are filled by the representations of $x_1, ..., x_n$, respectively. So `wft2` is a frame whose `member` slot contains the *filler* `Sufian`, and whose `class` slot contains the filler `Person`. In this example, `Call` is associated with a frame whose slots are `Communicator`, `Addressee`, and `Communication` (eg. `wft1`), `LocationOf` is associated with a frame whose slots are `Resident` and `Location` (eg. `wft3`), and `SourceOf` is associated with a frame whose slots are `Source` and `Theme` (eg. `wft6`).[2]

Caseframes, motivated by Fillmore's case theory [3], are comparable to relational database schemas, with slot names corresponding to attributes, and frames to rows of a relational database table. There are several differences, however: a SNePS frame slot may be filled with another frame; and a SNePS frame slot may have multiple fillers. Multiple fillers of a slot are considered to form a set (neither order nor multiplicity are significant), and are interpreted conjunctively. If one frame has two fillers in one slot and three in another, the frame is semantically equivalent to six frames with only one filler in each of the two slots.[3]

So far, we have discussed caseframes that are each associated with a single function symbol. However, there are times when using a different caseframe for each function symbol seems overly profligate. It is important to be able to derive a unique function symbol for each frame, both to create the logical expression corresponding to the frame, and to create a collapsed version, as discussed in Sect. 6. Therefore, to use a single caseframe for a set of related function symbols, we store the correct function symbol in one of the slots of each frame instance of the caseframe.

For example, consider a small KB containing the dependency parse (*see* [9]) for a single sentence. In a dependency parse, each clause and phrase is represented as a head word with various dependency relations to the other words and phrases in it. Thus, a dependency parse is a set of relationships of the sort *Dependency relation `drel` holds between head word `head` and dependent word `dep`*. Each of these can be represented as an instance of the caseframe with the slots `drel`, `head`, and `dep`, but we could write it as a logical expression using the filler of the `drel` slot as the function symbol, the filler of the `head` slot as the first argument, and the filler of the `dep` slot as the second argument.

To define such a caseframe, the user opens the pull-down menu under the "Name" button of the caseframe-definition dialog box, and selects "Function Symbols", as shown in Fig. 6. In this case, dependency relations used by the Stanford Typed Dependency Parser [10] have been typed into the "Function Symbols" box, specifically `amod`, `aux`, `ccomp`, `complm`, `det`, `expl`, `nn`, `nsubj`, `pobj`, and `prep`, and the slots `drel`, `head`, and `dep` have been specified, in that order. This indicates that the dependency

[2] The `Call`, `LocationOf` and `SourceOf` frames are based on the Contacting, Residence, and Source_of_getting frames of FrameNet [4,15], but simplified for this paper.

[3] This actually depends on the definitions of the slots, but is the default situation, and the only one we will consider in this paper.

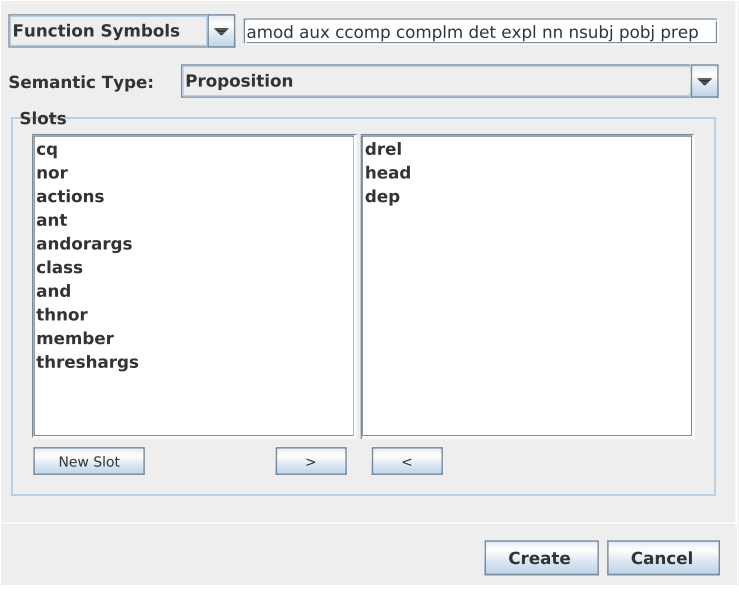

Fig. 6. Defining a caseframe which has multiple function symbols

relation whose logical expression is `(nsubj called Sufian)` (there is a clause whose main verb is "called" and whose subject is "Sufian") is to be represented by a frame whose `drel` slot contains `nsubj`, whose `head` slot contains `called`, and whose `dep` slot contains `Sufian`.

The name of the first slot, the one holding the function symbol, is used as the name of a caseframe that is associated with multiple function symbols. In the example shown in Fig. 6, the name of the caseframe being defined will be `drel`.

2.3 Propositional Graphs

A SNePS 3 KB may be conceived of as a graph. Every term, whether an individual constant, an arbitrary or indefinite term,[4] or a functional term (whether or not denoting a proposition), corresponds to a node in the graph. Slot names label directed arcs that go from nodes corresponding to frames (functional terms) to the nodes that correspond to the fillers of that slot in that frame (arguments of the function). This is illustrated in Fig. 7, which shows the same KB created in Fig. 2. Recall that an exclamation mark, "`!`", is appended to the `wft` names indicating those propositions that are asserted in the KB. For example, `wft6` is asserted to indicate that it is true that Ahmed was the source of the information that Ziyad is in Ramadi, but `wft5` is not asserted, indicating that Ahmed's information is not (yet?) believed. The graph is rendered within the GUI by the JUNG (Java Universal Network/Graph) system [22]. When the user rolls the cursor over a node, it's `wft` name and the logical expression corresponding to it are shown in the status line, as illustrated in Fig. 7.

[4] Arbitrary and indefinite terms are not illustrated in this paper.

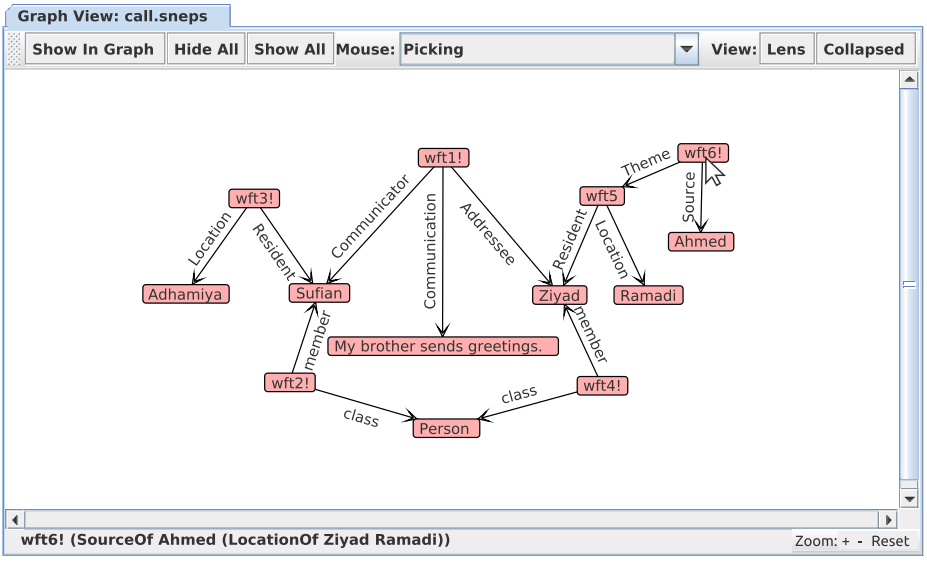

Fig. 7. A graphical view of the SNePS 3 KB created in Fig. 2, meaning *"Sufian, a person in Adhamiya, called Ziyad, a person who, according to Ahmed, is in Ramadi, saying 'My brother sends greetings.'"* The cursor is on the node labeled `wft6!`, so the `wft` name and the logical expression corresponding to this node are shown in the status line.

Frames associated with multiple function symbols are converted to propositional graphs using the same rules as frames associated with single function symbols, only the function symbol itself is now a slot filler, and so appears as a node in the graph. For example, Fig. 8 shows the dependency relations in the sentence *There is a large protest at the courthouse* using the same dependency relations shown as function symbols in Fig. 6. In the graph each frame is represented as a `wft` node with three arcs labeled `drel`, `head`, and `dep` going to the three slot fillers.[5]

A SNePS 3 graph may also be considered a hypergraph with labeled nodes and edges. The `wft` nodes correspond to hyperedges, with the benefit, however, that they can have arcs pointing into them (for example, `wft5` of Fig. 7). We believe that visualizing the graphs as in Figs. 7 and 8 is clearer than replacing the `wft` nodes with contours. However, a simplification is discussed and illustrated in Sect. 6 below.

3 Frame-Based Interaction

The addition of new knowledge to the KB requires that all slots of the desired caseframe are filled with the proper number of fillers, specified during the definition of the slots themselves. This number can vary between instances of a single caseframe. This would be difficult to enforce if new knowledge were added by drawing in the Graph View

[5] We actually use word tokens, with relationships to their word types, to accommodate sentences with multiple occurrences of word types.

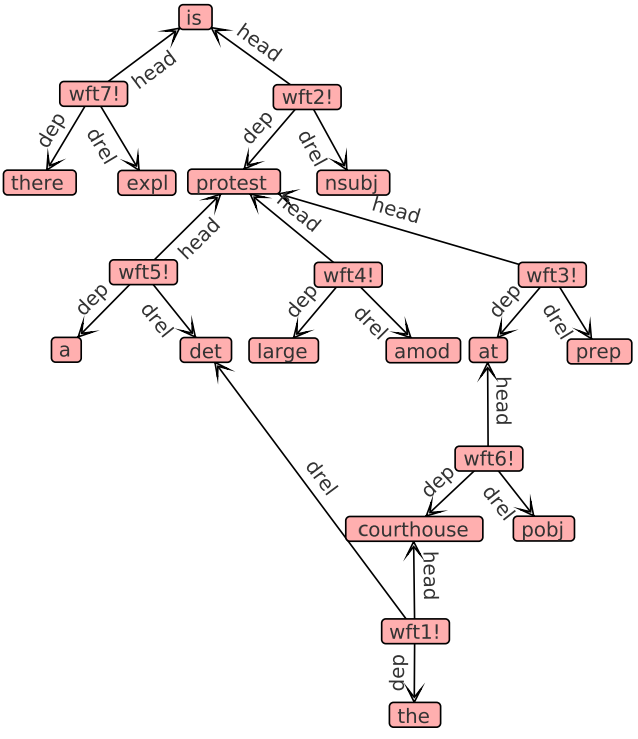

Fig. 8. The propositional graph for the dependency tree of *"There is a large protest at the courthouse"*

pane; the caseframe would have to be drawn one piece at a time, leading to periods where the graph being edited would be syntactically invalid and inconsistent with the KB that supports it. Keeping the user informed as to the ways the graph is incorrect and how the issues should be resolved is difficult. Our solution to this problem is for the user to add knowledge using either a frame-based interface or the traditional logic-based interface. In the logical interface discussed in Sect. 2.1, information is added to the KB using logical representations of frames. In fact, when terms are asserted or defined in the KB using the frame-based user interface, they are converted to the logical language first and displayed in the logical interface portion of the GUI.

The frame-based interface is illustrated in Fig. 9. Users select the caseframe which they wish to add an instance of to the KB. This automatically provides a frame-based view of the slots which require fillers. Conceptually, frames are often seen as tables of slots and fillers, and this is the metaphor we use here. As previously mentioned, the slots in a SNePS 3 frame may be filled by more than one term. For example, a single instance of the Isa caseframe can be used to say that both Toto and Fido are instances of both Dog and Pet. The interface shows this by showing multiple slot-filler rows with the same slot name. If users want to add additional fillers to a slot, they select the slot, add it, and type in the additional filler. The user also specifies whether the new frame is

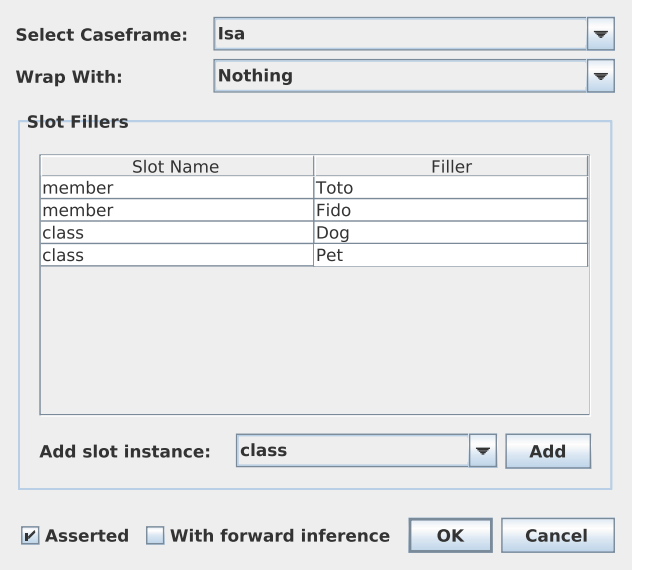

Fig. 9. A frame-based interface for assertions to the KB, showing multiple fillers for each of two slots

to be asserted, and, if so, whether to perform forward inference on it. The user may also specify a "wrapper" for the term definition, which is used for defining arbitraries and indefinities, not discussed here. In using an interface such as this, the inputs required by the user become obvious and easily machine verifiable. No changes to the graph, and hence KB, are made until the change is guaranteed to result in a valid assertion to the KB.

Queries can be performed on the KB and the results displayed in the graph using Query By Example [23]. This is the same as the visual QBE interface designed for database systems wherein the user is shown an empty row for a table and can enter values into fields they are interested in in order to formulate a query. This speaks to the general similarity of a KB with a database system wherein frames are like rows in a database table and the caseframe defines the table itself.[6]

Using the QBE interface, as shown in Fig. 10, users first choose the caseframe they are interested in, and enter fillers for none, some, or all of the slots shown. All slots not filled will be converted to variables $\{?w_1, .., ?w_n\}$. The system converts the text entered into the logical language and queries the KB. The resulting matches are displayed in the graph. If previously the entire KB was displayed in the Graph View pane this is treated as a filter, and the rest of the KB is hidden. Otherwise, the results are added to the Graph View pane.

[6] The analogy begins to break down when we consider two abilities which differentiate a caseframe from a database table: the ability to have multiple fillers of a slot, and the ability to fill a slot with an instance of another frame.

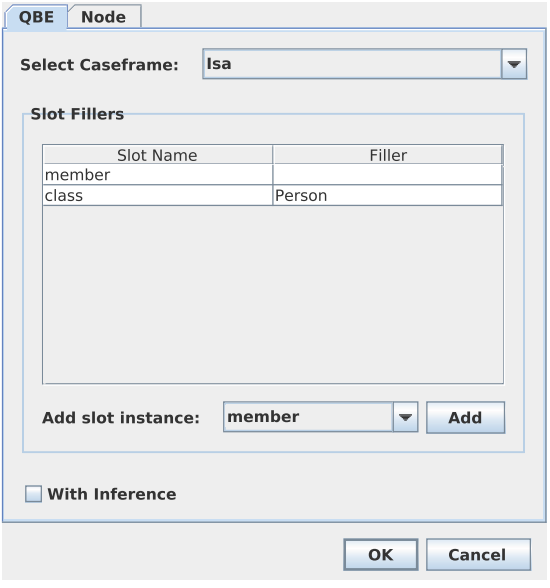

Fig. 10. The QBE dialog box indicating a query for instances of the `Isa` caseframe with the class `Person`

In this example, the user is interested in instances of the `Isa` caseframe in which the `class` slot has the filler `Person`. We will apply this frame-based query to the KB containing the knowledge about the call Sufian made to Ziyad discussed earlier. Since the entire graph was shown before the query, it's assumed the user means to show only the results of the query. The system queries the KB for matches to the query and displays only those on the graph, as seen in Fig. 11. Inference is not performed in querying the KB unless the appropriate check box is ticked. This graph contains only the graphical representations of `Isa` frames in which the slot for `class` has the filler `Person`. We could now continue querying in this way to add relations to the graph until we have what is useful to us. It's also possible to add a single node (or list of nodes) to the graph rather than a relation using the interface shown in Fig. 12. The user interface enforces the notion that the graph visible to the user should be syntactically valid at all times, and will display a `wft` node only with all edges out of it visible.

Users may explicitly use variables in their query by pre-pending a question mark, "`?`", to the variable symbol, a feature useful for querying metaknowledge. For example, a user may view the graph with all instances of the `SourceOf` caseframe where the `Source` slot contains `Ahmed` and the `Theme` slot contains `(LocationOf Ziyad ?x)` to display all relations representing locations Ahmed has said Ziyad has been. The user can enter this portion of the query either manually by typing it in to the appropriate filler cell in the table, or by double clicking the cell to display a dialog for specifying the nested term, as shown in Fig. 13.

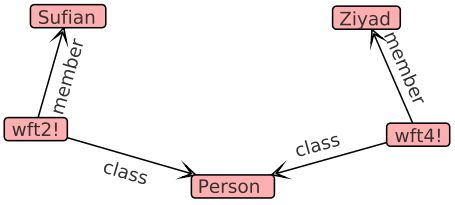

Fig. 11. The result of finding instances of the `Isa` caseframe with the class `Person` in the calling KB shown in Fig. 7

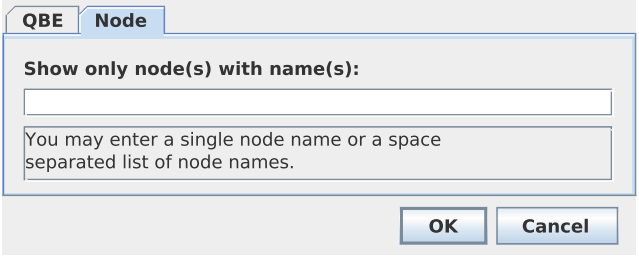

Fig. 12. An interface to display a single node or list of nodes in the graph

4 Graph-Based Interaction

At any time, the Graph View pane may be showing: the entire KB graph, after the user clicked on "Show All"; nothing, after the user clicked on "Hide All"; selected sub-graphs, after using the QBE interface; or even isolated base nodes (nodes with out-degree 0), after using the "Node" tab of the QBE dialog box. The one constraint, however, is that whenever a `wft` node is displayed, so are all its outgoing arcs and the nodes they go to, because otherwise, the displayed graph would not be syntactically correct.

When some nodes are shown in the Graph View pane, the user may explore the KB graphically by right-clicking on a node. A context-sensitive popup menu then appears as shown in the left-most part of Fig. 16 with the following choices, but only those that are relevant.

Show All In Edges (n edges). All the in edges are shown, along with the `wft` nodes they come from, their out edges, etc. n is the number of in edges that exist in the KB, but are not currently shown.

Show In Edges By Relation A dialog box appears which lists all the caseframes of frames this node is currently a filler in, but which are not currently shown. The user can select any number of caseframes, and the corresponding `wft` nodes are shown.

Hide All In Edges (n edges). All the in edges are hidden, along with the `wft` nodes they come from, their out edges, their in edges, etc. n is the number of in edges currently shown.

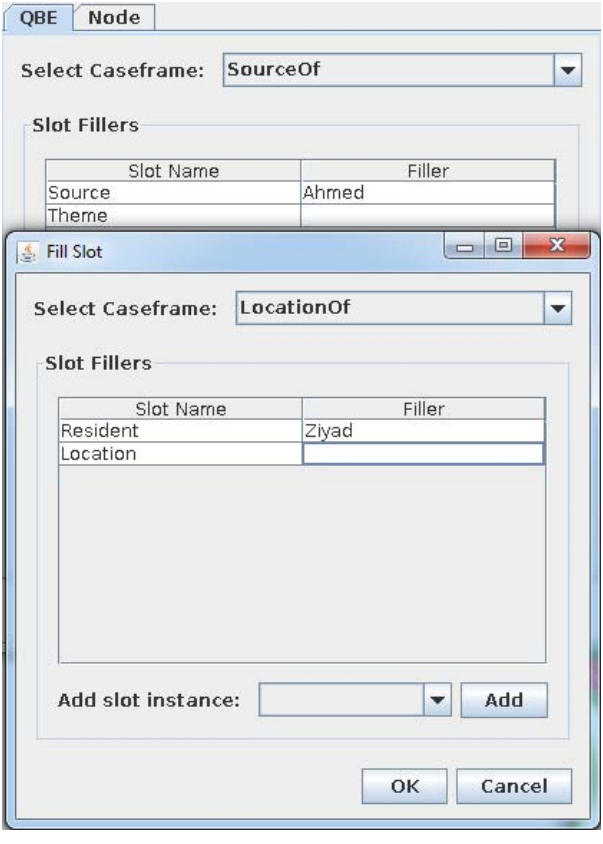

Fig. 13. The user interface to fill a slot with a frame. The user has double clicked in the Filler field of the `Theme` slot.

Hide In Edges By Relation. A dialog box appears which lists all the caseframes of frames this node is currently a filler in, and which are currently shown. The user can select any number of caseframes, and the corresponding `wft` nodes are hidden.

Hide Node. The node is hidden, along with all its out edges, all its in edges, the `wft` nodes the in edges come from, etc.

Consider the example shown in Fig. 11. If we were interested in all the knowledge about Sufian, we could right-click on the node `Sufian`, select "Show All In Edges", and obtain the graph in Fig. 14. If we were only interested in some of the knowledge attached to `Sufian` we could look at the status bar when mousing over the `Sufian` node to notice that it is an argument in the `Call`, `LocationOf`, and `Isa` relations, as shown in Fig. 15. If we were only interested in knowledge attached to `Sufian` in `Call` relations we could then use the process shown in Fig. 16. In this way the user can begin with a QBE-based query and only expand nodes of interest.

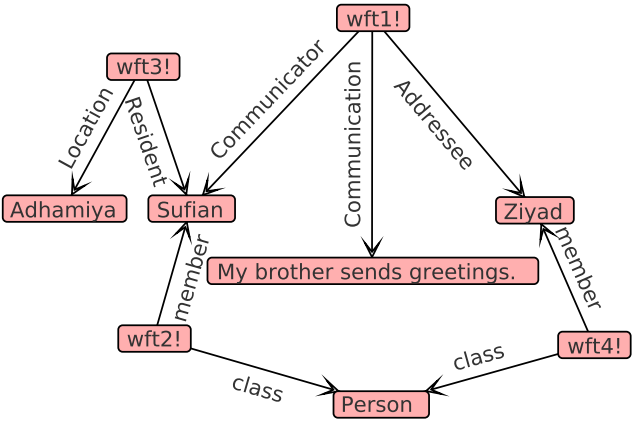

Fig. 14. An expansion of the graph shown in Fig. 11 showing all the relations attached to the node `Sufian`

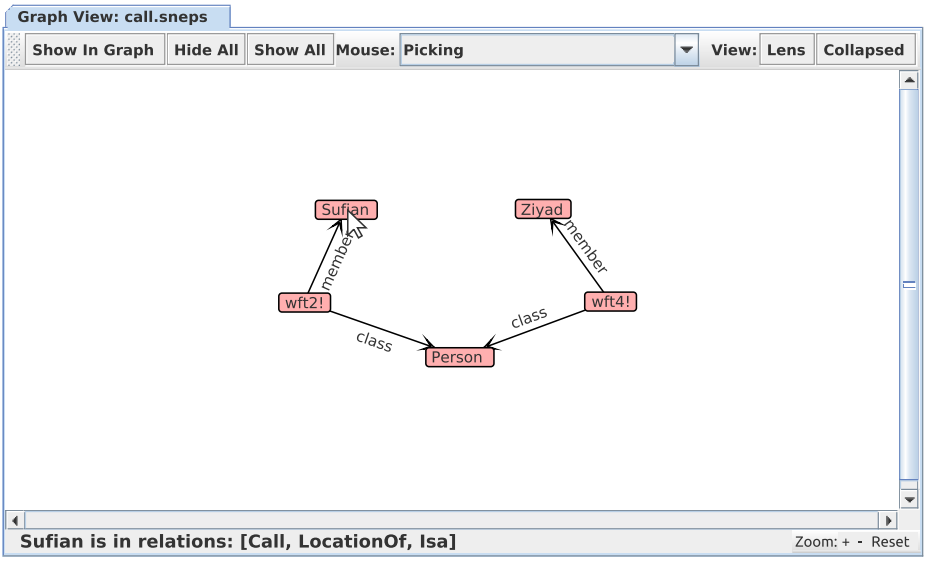

Fig. 15. When the cursor is positioned over a node, the status bar shows the relations (caseframes) the node is an argument (a filler) in. In this case it shows that `Sufian` is an argument in the `Call`, `Isa`, and `LocationOf` relations.

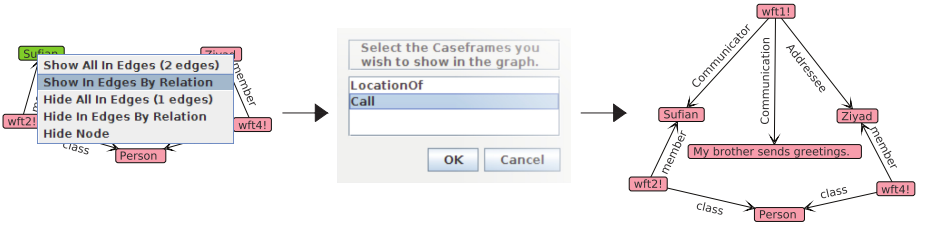

Fig. 16. The process to expand only the `Call` relation attached to `Sufian` in the graph shown in Fig. 15. The user right-clicks on the `Sufian` node, selects "Show In Edges By Relation", selects "Call", and clicks "OK" to get the graph shown on the right.

This example is of course quite trivial given the size of the graph in Fig. 7. In a much larger KB this becomes extremely useful as a method to visualize specific portions of a KB.

5 Use for a Large Knowledge Base

In order to see the usefulness of QBE on a larger scale, we consider a knowledge base from a current project on soft information fusion [12,13] containing 2,075 terms consisting of place names and various pieces of information about each place from the NGA GEOnet Names Server [11]. When these terms are added to the knowledge base and visualized, the graph shown in Fig. 17 is produced. Large KBs are very difficult to work with conceptually as there is a significant burden on the user to understand the complex structure. The purpose of visualizing a KB should be to give the user a way to see the parts of the KB which are interesting to them, while reducing this burden as much as possible. As you can see, visualizing an unmodified graphical version of the entire KB is unhelpful - it's impossible to pick out even a single relation.

Very rarely is a user or knowledge engineer really interested in the entire KB at one time, so constraining the display would be useful. Assume we are really only concerned with places that are countries, and we'd like to see the nodes for all of those. We can do this with a simple QBE frame-based query (by selecting the `Isa` caseframe, entering `Country` in the Filler field of the `class` slot, and leaving the Filler field of the `member` slot empty). The graph is then restricted to showing only the relevant knowledge, as shown in Fig. 18.

As you can see the resulting graph is much easier to understand; an underlying structure is visible and a node of interest is easier to pick out. The user could easily find and expand a country node if she were interested in the relations attached to it. One potential problem with this view is that the propositional graph contains many `wft` nodes which may not be helpful to the user of the graph. Under some situations these can be removed by redrawing, or collapsing the graph.

6 Collapsing the Graph

It is important to display the `wft` nodes in the graph either if the function has more than two arguments, as is the case for `wft1` in Fig. 7, or if the term is, itself, an argument of

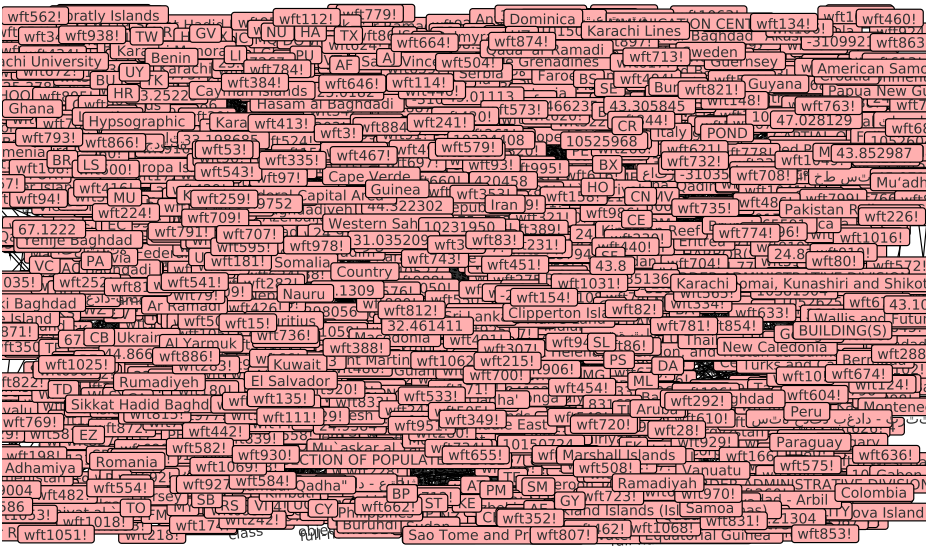

Fig. 17. A large network of places and knowledge about them from the NGA GEOnet Names Server

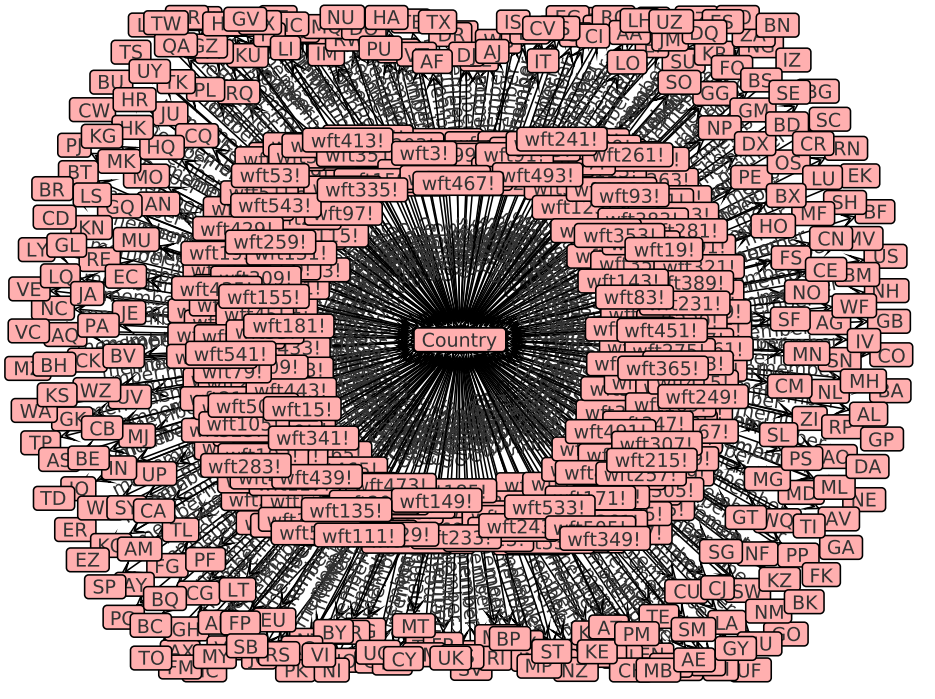

Fig. 18. The KB shown in Fig. 17 filtered by QBE to show only places that are countries

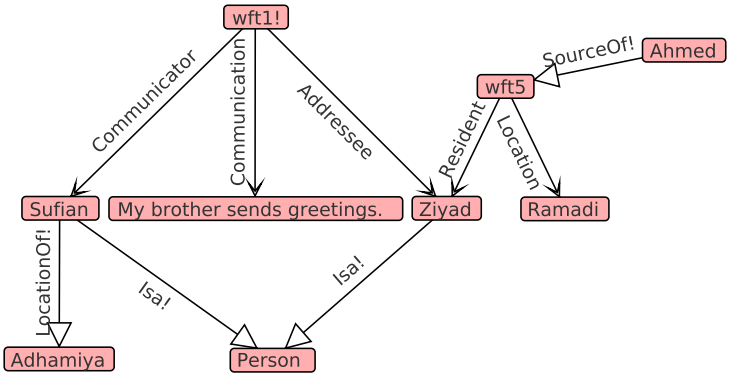

Fig. 19. A collapsed version of the graph shown in Fig. 7 in which binary relations are redrawn without their `wft` nodes

another function, as is the case for `wft5`. However, in the case of functional terms that have just two slots and that are not fillers of other slots, such as `wft2`, `wft3`, `wft4` and `wft6`, the `wft` nodes just clutter the display. In this case, the `wft` node with its two outgoing arcs can be converted into a single arc labeled by the function symbol. This is referred to as "collapsing" the graph. The resulting collapsed graph is semantically equivalent to the uncollapsed version, it is only an alternate way of visualizing the knowledge in the KB. Figure 19 is a collapsed version the graph shown in Fig. 7.[7]

Because the graph representation is backed by frames with ordered slots, the GUI can automatically determine which parts of the graph can be collapsed, the direction of the new arcs, and their labels. Any frame with only two slots and only one filler per slot can be collapsed. The arc is drawn from the filler of the first slot to the filler of the second slot, and the label of the new arc is the function symbol associated with the frame (the frame's name). If the new collapsed arc replaces a `wft` node which is asserted, the function symbol has an exclamation point, "`!`", appended to it. The arrow head used on the collapsed arcs is a different style from that used on uncollapsed arcs as a visual reminder to the user that it is a collapsed arc.

In the case of a caseframe associated with more than one function symbol, if there are three slots, and only one filler in each, then the collapsed arc is drawn from the filler of the second slot to the filler of the third slot, and labeled with the filler of the first slot, the frame's name. Figure 20 shows the collapsed version of the dependency graph shown in Fig. 8.

Notice that, in contrast to some other systems, for example those assumed by [1], not all SNePS 3 `wft` nodes can be collapsed in this way. This is due to the increased expressibility of SNePS 3: the ability to represent n-ary relations for $n > 2$, and the ability to express propositions about propositions (and about other functional terms).

We can apply this to the NGA GEOnet KB to which we applied a QBE filter earlier (shown in Fig. 18) to increase the usability of that graph further. In collapsing this graph,

[7] An inspiration for collapsing the graph in this way was NETL's handle nodes [2].

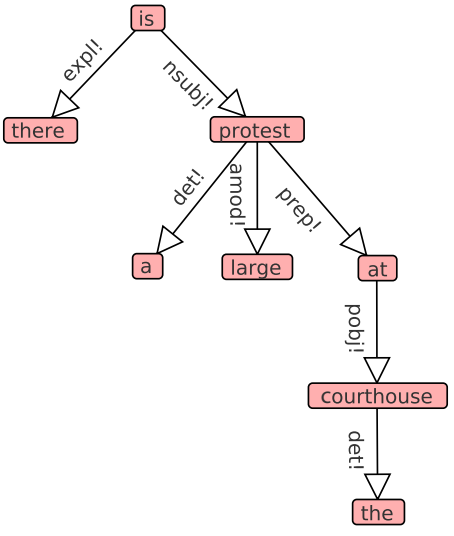

Fig. 20. The collapsed version of the graph shown in Fig. 8

the binary relation `Isa` is visualized by only two nodes instead of three as shown in Fig. 21 and the graph is now easier to view and manipulate.

As you can see, this increases the usability of the graph interface even beyond that of using QBE alone, allowing for very quick visualization of important data from complex hierarchies. These techniques together are more useful than automatic clustering of entire sections of a graph or using a zooming function. Automatic clustering can change the semantics of the graph and hide important knowledge and structural features. Scaling a graph on the other hand does not change the semantics, but it still hides relevant parts of the graph along with irrelevant parts, degrading the users experience.

7 Evaluation

The user interface discussed here has been used for visualizing and interacting with graphs on the order of several thousand nodes. The techniques are known to scale to this level, and we are confident they will scale beyond it as work continues with increasingly larger knowledge bases. Any limitations in scale we believe will be due to the JUNG visualization system rather than the techniques discussed here. Unfortunately we were unable to find user interfaces for generalized KR systems suitable for comparison with the SNePS 3 user interface. Instead, we will discuss the concepts developed here in the context of ontology editors since they face many of the same challenges discussed here. We looked at several ontology editors [6,14,21] and found that all of them provide the same basic functionality in mostly similar ways, so we will discuss this in the context of Protégé, which appears to be the most popular.

The increased expressivity of the SNePS 3 logic differentiates this work from ontology editing tools. Ontologies limit themselves to the use of binary relations to show

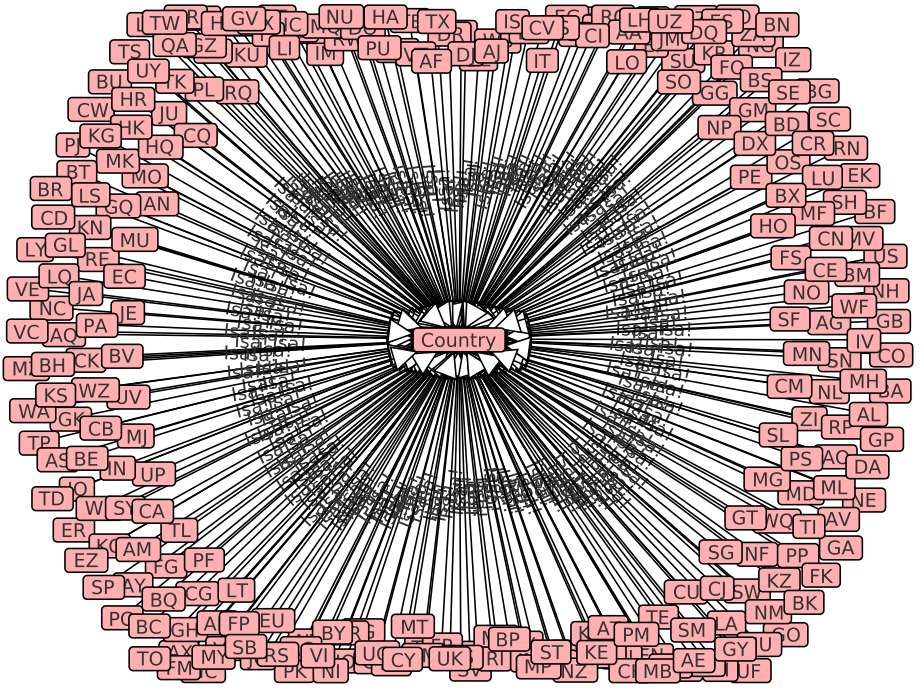

Fig. 21. A collapsed version of the graph shown in Fig. 18 to show places which are countries with binary relations collapsed

the relationships between classes and individuals. In Protégé, to add a restriction between two classes, one needs only choose the initial class, the restricted property (such as `is-a` or `part-of`), and the restricted filter (the second argument in the binary relation). The interface is very simple for this—the user chooses from a tree of options. The frame-based interface for adding an instance of a caseframe we use allows for n-ary relations and adding instances of frames as the values for slots, requiring a more complex interface. For purely binary relations the work done in the Protégé interface works well, but it does not scale beyond that.

The methods of interaction with an ontology in Protégé are somewhat similar to what we have discussed here. The addition of classes and instances to an ontology occurs outside of any graph interface, and the graphs are used for visualization purposes only. The ontology editors surveyed also have basic reasoning abilities and the ability to present a query to the reasoner. The SNePS 3 user interface allows the user to pose a query to the system, and specify whether or not inference should be used. If inference is not used, QBE can be seen as a way to display a filtered version of the knowledge base. When inference is used, it invokes the SNePS 3 reasoner[8] to display all of the results of the query.

[8] Currently, the SNePS 3 reasoner supports forward and backward chaining on ground formulas with set-oriented connectives [19] using a natural deduction proof system. A full reasoner using the logic containing arbitrary and indefinite terms [18] is being designed.

The visualization requirements of a KR system and an ontology viewer are somewhat different. Ontologies always have a defined root and lend themselves to a hierarchical tree-like view, while this is not always the case in a propositional graph. This characteristic leads to ontology editors which show a definable number of levels of the graph in detail, along with some number of nodes along the path to the root (as in Protégé's OWLViz viewer). The graphs are also always shown with a view akin to what we call the "Collapsed View" in the SNePS 3 interface, as there are only binary relations to be shown. These features in combination allow ontologies which are very large (containing millions of concepts) to be viewed easily. In order to view very large KBs, we require the user to query the KB.

SNePS 3 and its user interface can be (and have been [13]) used for interaction with ontology structures, providing for the most part a superset of the visualization and interaction features provided by ontology editors.

8 Conclusions

The nature of SNePS 3 KBs being conceived of as simultaneously graph-, frame-, and logic-based extends naturally into the interactions one may do graphically with such KBs. Users are able to add knowledge to the KB and visualize it in a user-friendly way. The combination of frame-based techniques for QBE, and graph-based techniques for exploration has resulted in a highly usable method for interacting with KBs. The `wft` nodes are important for n-ary relations when $n > 2$, and when they are functional terms that are arguments in other functional terms, yet they may be collapsed when not needed, in order to provide a simpler graph for the users to look at without sacrificing semantics. It is now possible to view, modify, and understand a large KB without the need to sift through complex text outputs or graphs which are too large to be viewed easily. We believe the techniques given here are ideal for visualizing and interacting with the data associated with a KR system.

Acknowledgments. We thank the SNePS Implementation Group for their feedback on the user interface. This work has been supported in part by a Multidisciplinary University Research Initiative (MURI) grant (Number W911NF-09-1-0392) for "Unified Research on Network-based Hard/Soft Information Fusion", issued by the US Army Research Office (ARO) under the program management of Dr. John Lavery.

References

1. Bodenreider, O.: Experiences in visualizing and navigating biomedical ontologies and knowledge bases. In: Proceedings of the ISMB 2002 SIG Meeting: Bio-ontologies, pp. 29–32 (2002)
2. Fahlman, S.: NETL: A System for Representing and Using Real-World Knowledge. MIT Press, Cambridge (1979)
3. Fillmore, C.J.: The case for case. In: Bach, E., Harms, R.T. (eds.) Universals in Linguistic Theory, pp. 1–88. Holt, Rinehart and Winston, New York (1968)

4. Fillmore, C.J.: Frame semantics and the nature of language. In: Annals of the New York Academy of Sciences: Conference on the Origin and Development of Language and Speech, vol. 280, pp. 20–32 (1976)
5. Franz Inc., Oakland, CA: Allegro CL 8.2 Documentation (2010), http://www.franz.com/support/documentation/8.2/doc/
6. Harris, N.: OBO-Edit (2011), http://oboedit.org/
7. Hickey, R.: Clojure (2011), http://clojure.org/
8. ISO/IEC: Information technology — Common Logic (CL): a framework for a family of logic-based languages, ISO/IEC 24707:2007(E). ISO/IEC, Switzerland, 1st edn. (October 2007), http://standards.iso/ittf/license.html
9. Jurafsky, D., Martin, J.H.: Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition, 2nd edn. Prentice Hall, Upper Saddle River (2000)
10. de Marneffe, M.C., Manning, C.D.: Stanford typed dependencies manual. Stanford Natural Language Processing Group (September 2008), http://nlp.stanford.edu/software/dependencies_manual.pdf
11. National Geospatial-Intelligence Agency: NGA GEOnet names server, GNS (2011), http://earth-info.nga.mil/gns/html/
12. Prentice, M., Kandefer, M., Shapiro, S.C.: Tractor: A framework for soft information fusion. In: Proceedings of the 13th International Conference on Information Fusion, Chicago, IL (2010)
13. Prentice, M., Shapiro, S.C.: Using propositional graphs for soft information fusion. In: Proceedings of the 14th International Conference on Information Fusion, Edinburgh, UK (2011)
14. Revelytix, Inc.: Knoodl (2011), http://www.knoodl.com/
15. Ruppenhofer, J., Ellsworth, M., Petruck, M.R.L., Johnson, C.R., Scheczyk, J.: FrameNet II: Extended theory and practice (2006) (unpublished manuscript)
16. Shapiro, S.C.: Belief spaces as sets of propositions. Journal of Experimental and Theoretical Artificial Intelligence (JETAI) 5(2&3), 225–235 (1993)
17. Shapiro, S.C.: An Introduction to SNePS 3. In: Ganter, B., Mineau, G.W. (eds.) ICCS 2000. LNCS(LNAI), vol. 1867, pp. 510–524. Springer, Heidelberg (2000)
18. Shapiro, S.C.: A logic of arbitrary and indefinite objects. In: Dubois, D., Welty, C., Williams, M. (eds.) Principles of Knowledge Representation and Reasoning: Proceedings of the Ninth International Conference (KR 2004), pp. 565–575. AAAI Press, Menlo Park (2004)
19. Shapiro, S.C.: Set-oriented logical connectives: Syntax and semantics. In: Lin, F., Sattler, U., Truszczynski, M. (eds.) Proceedings of the Twelfth International Conference on the Principles of Knowledge Representation and Reasoning (KR 2010), pp. 593–595. AAAI Press, Menlo Park (2010)
20. Shapiro, S.C., Rapaport, W.J., Kandefer, M., Johnson, F.L., Goldfain, A.: Metacognition in SNePS. AI Magazine 28, 17–31 (2007)
21. Stanford Center for Biomedical Informatics Research: The Protégé ontology editor and knowledge acquisition system (2011), http://protege.stanford.edu/
22. The JUNG Framework Development Team: JUNG - Java universal network/graph framework (2010), http://jung.sourceforge.net/
23. Zloof, M.M.: Query by example. In: AFIPS, vol. 44 (1975)