A Demonstration of KSIMapper and Constraint Graphs

Rob Kremer
Knowledge Science Institute
University of Calgary
Calgary, Alberta Canada T2N 1N4
kremer@cpsc.ucalgary.ca

Abstract

This paper describes a demonstration of KSIMapper and Constraint Graphs. Both these programs are described in the accompanying paper "Toward a Multi-User, Programmable Web Concept Mapping "Shell" to Handle Multiple Formalisms" (Kremer 96a) in the KAW'96 proceedings.

KSIMAPPER

KSIMapper is the simplest of this collection of concept mapping programs. It is merely a demonstration program for the CMap (Kremer 96b) class library on which all the other programs are built. There are no semantics imposed on the graph whatsoever. It is not much more than a specialized drawing program.

Its basic properties are:

- Graphs consist of three kinds of objects: *nodes*, *arcs*, and *contexts*.
- Nodes may or may not have surrounds, which may be of arbitrary shape, fill color, surround color, and text color.
- Arc terminals can "float" or be "pinned" to other objects (node, contexts, or other arcs).
- Arc terminals may be decorated with any number of "arrowheads" from an extendible library of shapes. Figure 1 shows simple black arrowheads as well as red arrowheads; some terminals have no arrowheads at all.
- Arcs are n-ary: they can be unary, binary, or of higher arity. The *BTWN* arc in Figure 1 is a trinary arc.
- Arcs may be labeled or unlabeled; labels may or may not have surrounds, which may be of arbitrary shape, fill color, surround color, and text color. Most of the arcs in Figure 1 are unlabeled, but the *IsA* arc has a label with no surround, while the *BTWN* and *AGNT* arcs have elliptical surrounds.
- Arcs labels may float (automatically move when objects

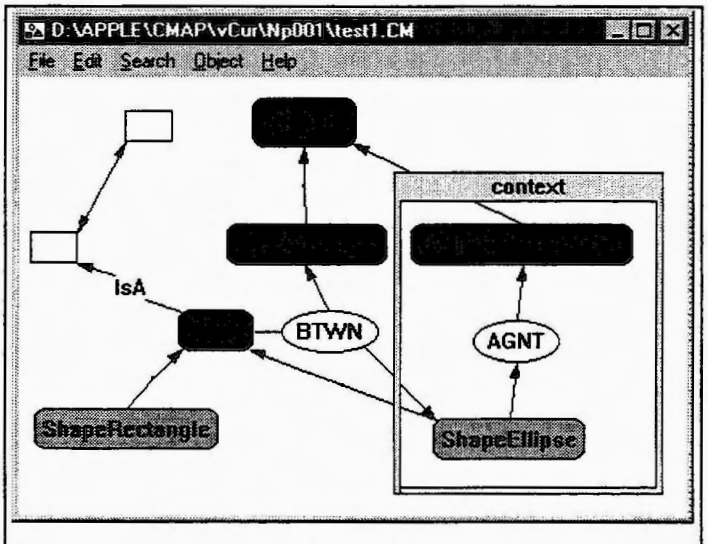

Figure 1: KSIMapper

at terminals are moved) or may be "pinned" to specific coordinates.

- Arc terminals may be re-attached by selecting them, then dragging them on top of new objects.
- Objects which are completely enclosed in a context remain enclosed in the context when the context is moved..
- All object types may be moved by dragging them around the drawing surface.

NPKSIMAPPER AS A LIVECONNECT PLUG-IN

NPKSIMapper is the Netscape plug-in version of KSIMapper. It uses the same source code as KSIMapper, but is extended as a plug-in. Figure 2 shows a NPKSIMapper embedded in an HTML document. This plug-in program uses LiveConnect (Netscape, 1996a) to masquerade as a Java class. The buttons along the bottom are not part of the plug-in, but are part of the HTML code in the document: part of an HTML form. They are scripted with JavaScript (Netscape, 1996b) to send Java messages to the plug-in. For example, the "save as" button is defined as follows:

<input type = "button" name = "save as" value = "save as" onClick = "document.map.saveAs()">

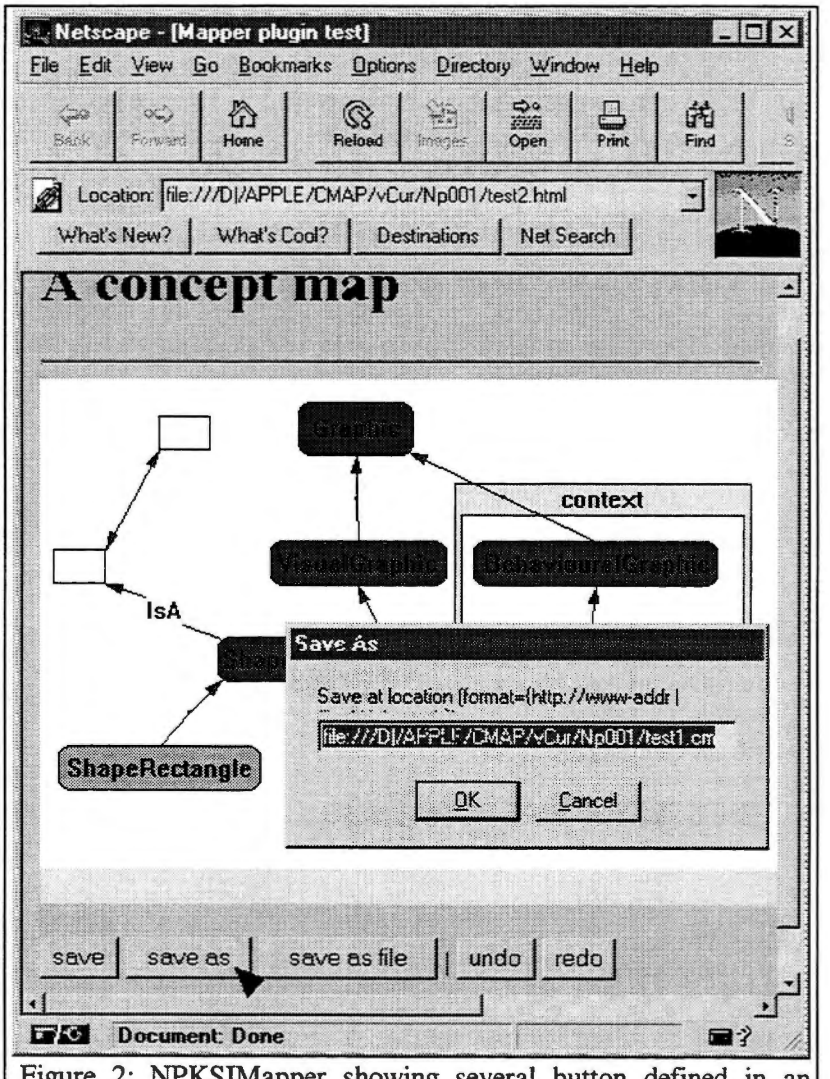

Figure 2: NPKSIMapper showing several button defined in an HTML form that interact with the NPKSIMapper Plug-in.

In Figure 2 the user has just clicked the "save as" button, which has sent the corresponding message to the plug-in, which has caused the plug-in to display its "Save As" dialog box.

NPKSIMapper can interact with other processes over the Web in real time, and can write back concept maps to the server. Remote communication with other processes is accomplished by using a CGI program and a second daemon process on the server.

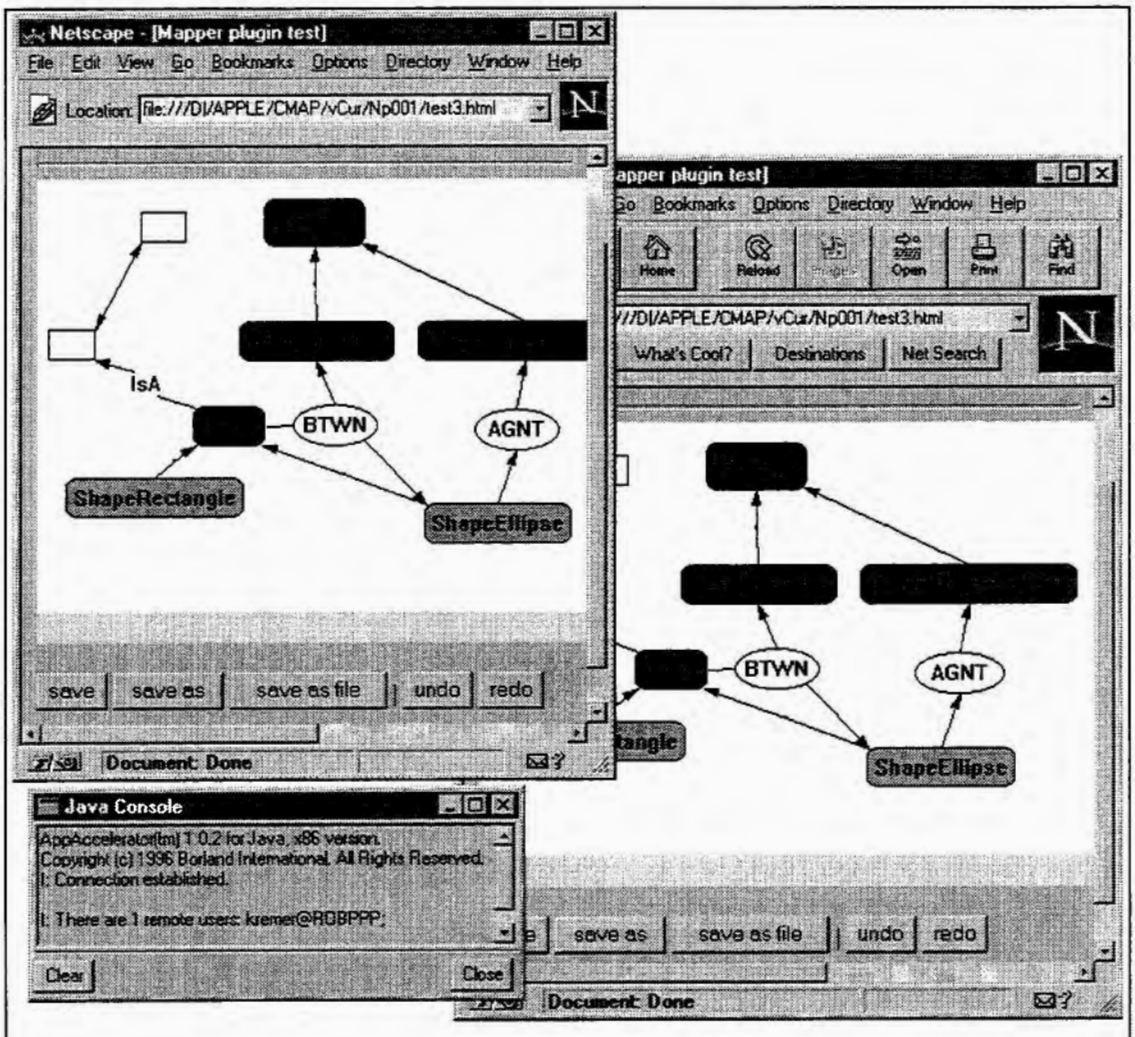

Figure 3: Two separate processes communicating via a server in a real-time interaction situation.

Figure 3 shows two separate Netscape processes viewing the same concept map located on the server. When the user performs any operation (e.g. moving a node) on either of these maps, the operation will be broadcast through the server to the other process, so that each of the processes will be maintained in the same state. There is no one process that "has the floor". Either process may save the modified map back to the server. There is no fundamental limit to the number of processes that can concurrently communicate in this way.

Since the plug-in is using LiveConnect, it is easy for it to use all the resources available to Java programs. Figure 3 shows the plug-in in the smaller Netscape window has logged several messages to the Java console.

CONSTRAINT GRAPHS

Constraint Graphs is a program that imposes user-definable syntax constraints on a graph. It uses the CMap class library for its interface, just as KSIMapper does.

Creating Node, Arcs, and Contexts

Figure 4 shows a typical menu the user has brought up by right-clicking the mouse on the background region of a concept map (every object has a menu available through the right mouse button — including the background). In this case, it is a map that has been specialized to emulate KDraw (Gaines, 1991). The user has chosen *New Node* from the menu and now may select the type of the node from a list of possibilities within the KDraw ontology of base types. These base types are node labels from other nodes that a system author has created (Figure 7) and then "made public" for the domain user (see Section 0). Arcs and Contexts may be created in a similar manner. (After creation, arcs may be attached to appropriate objects by dragging the end points.)

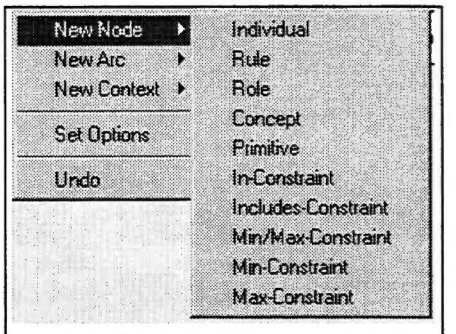

Figure 4: A typical background menu

Figure 5 shows a scenario where the user has just drawn an arc from a *Employee Concept* to a *Person Concept*. The arc is drawn by dragging the endpoint of an "elastic line" rooted in *Employee* and dropping it over *Person*. The program automatically searches for all possible legal typings for the new arc. In this case, there are three possibilities, so the system has displayed a menu for the user to select among them. If there had been only one possibility, the system would have just used it. If there had been none, the system would have displayed an error message. Just as in the case of nodes, above, the type possibilities are drawn from public symbols created by the system author in Figure 7.

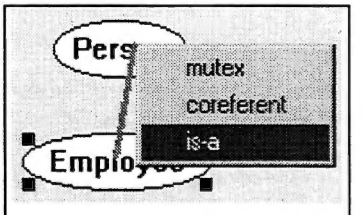

Figure 5: Using "elastic lines" to create an arc between two nodes

Adding Attributes to Objects

Figure 6 shows the dialog box the user can bring up via an object's menu. In this case the object is a subtype of an *isa* arc, so the dialog box shows information about ontology filters (more about ontology filters later); for non-*isa* arc objects, this section is not displayed. The attributes section is labeled with a combo box that allows the user to select an existing attribute or create a new attribute. The user may similarly select a type for the attribute; although, in this case, the type selection box has been grayed out because the attribute's type is either defined in a parent class or is a fixed by the system. The value field may be one of several tools, depending on the type of the attribute. In this case it is a selection from a pull-down list. The priority box allows the user the disambiguate inheritance in a multiple-

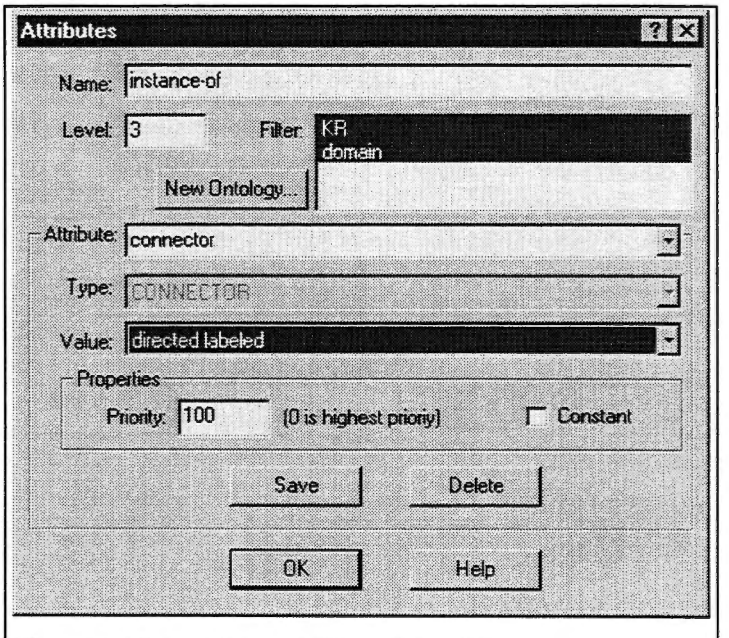

Figure 6: The attributes editing dialog box.

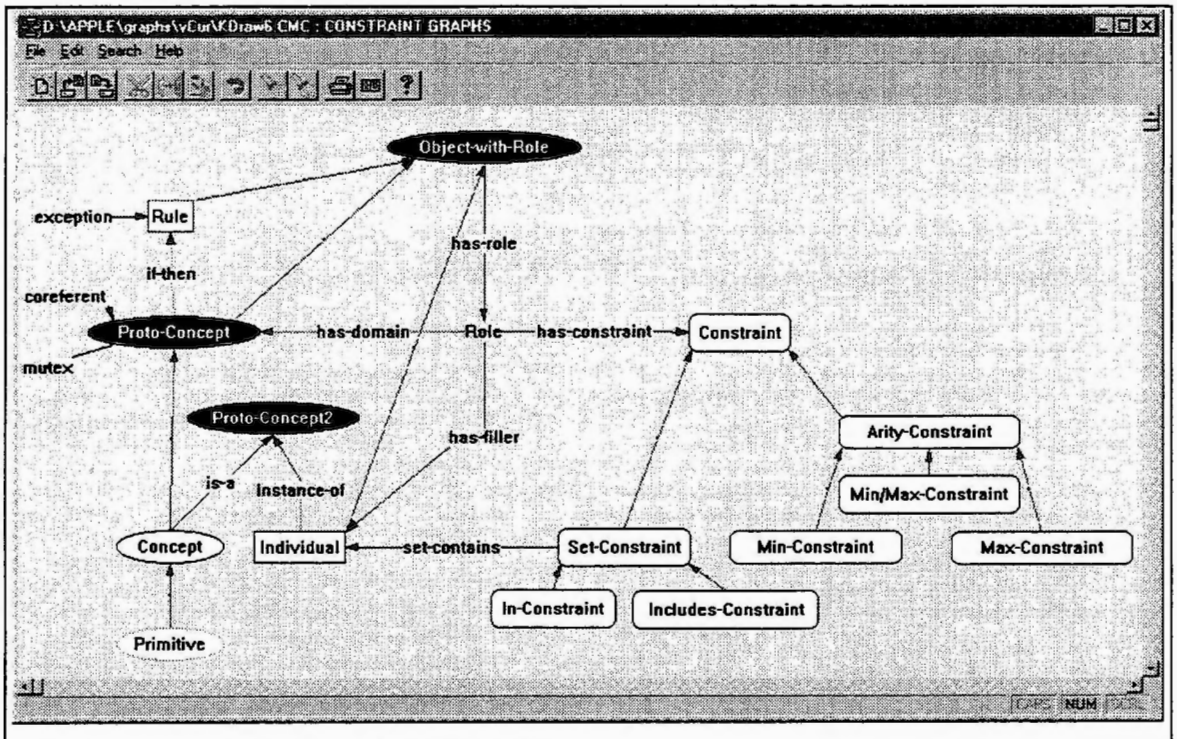

Figure 7: A description of the KDraw ontology.

inheritance situation, and the constant box may be checked to disallow overriding of the attribute and its value. The Save and Delete buttons may be grayed out if the attribute is not available for change (it's a constant in an ancestor class) or its value is not overridden by the current class, respectively.

Inheritance

Figure 7 is a Constrain Graphs description of the base types in KDraw. The lines with the red arrow heads are all *isa* arcs. The system considers any object at the root of an *isa* arc to be a subtype of the object at the head of the *isa* arc. *Isa* arcs may be applied to nodes, arcs (including *isa* arcs) and contexts. Several restrictions apply to *isa* arcs, such as "no cycles" and "arcs and nodes are mutually exclusive".

The system uses *isa*-defined inheritance for two purposes:

- to determine type legality: Any constraint that applies to an object must also hold for all its subtypes. In addition, all arcs require that objects at their terminals must be subtypes of the objects at the corresponding terminals of their supertypes. For example, Figure 7 dictates that only subtypes of *Individual* may be at the root of the *instance-of* arc, and only subtypes of *Proto-Concept2* may be at its head. *Instance-of* is a subtype of *isa* (but this is not shown directly in the description to avoid clutter).

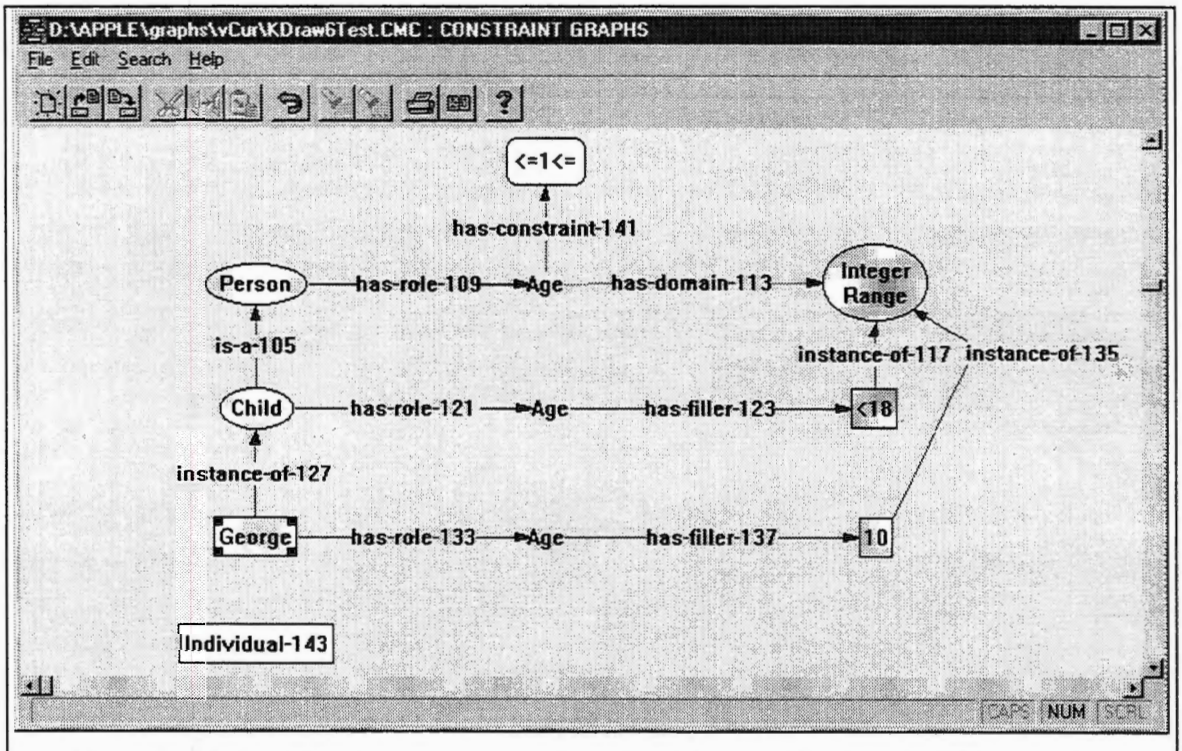

Figure 8: A domain description using the system description of KDraw shown in Figure 7.

- to infer visual appearance: Visual attributes such as surround shape, fill color, border color, line color, and text color are determined by inheriting from the attributes of the ancestor objects. Figure 8 shows that the user has colored *Person* yellow, and this attribute has been inherited by its subtypes, *Child* and *George*.

Levels and hiding

Figure 9 shows the Constraint Graphs options dialog box. This is generally used by the system author who is describing a base ontology (e.g. Figure 7) for later use by another user who will use it to create a specific domain (e.g. Figure 8). The fields are used as follows:

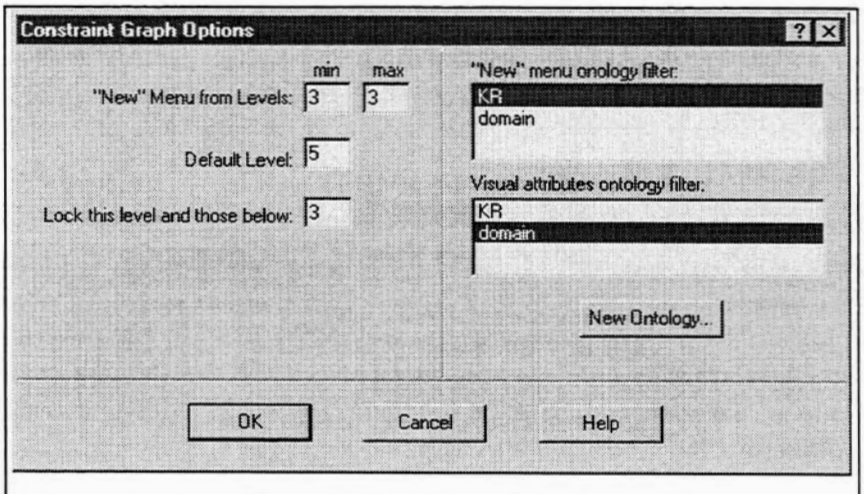

Figure 9: The options dialog box

- "'New' Menu from Levels" is used to constrain the system to only suggest types from specific levels in its various type-selection menus (e.g. Figure 4).
- "Default Level" is used to specify at which level to create new objects.

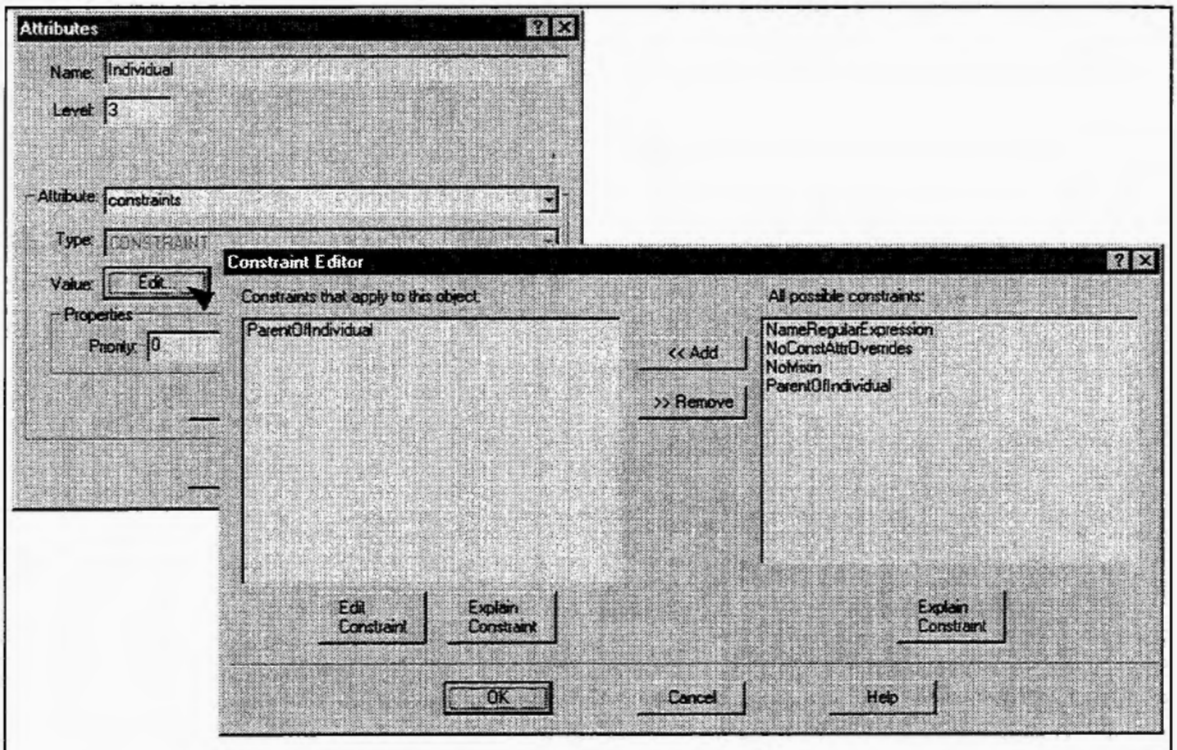

Figure 10: The constrain editor activated by the attributes dialog box.

- "Lock this level and those below" is used to prohibit editing at system levels.
- "'New' menu ontology filter" and "Visual attributes ontology filter" are used to create slightly different inheritance lattices for different purposes where necessary. This feature allows inheritance of visual attributes between *Concepts* and *Individuals* (as in Figure 8) without giving the *Individual* all the properties of *Concepts*. This problem also serves to explain some of the complexity of Figure 7.

Constraints (Predicates)

Constraints are predicates which are associated with objects. No operation is considered legal if any of its constraints (and those of all its ancestors) do not hold after the operation's application. Constraints are implemented as C++ function objects as per the Standard Template Library (Nelson, 1995).

Figure 10 shows the constraint editor which the user has activated by clicking the value field of the *constraints* attribute in the attribute dialog box. Users may select constraints from a library by selecting them from the list on the right. If a constraint has parameters, a dialog will automatically pop up to give the user a chance to edit the parameters. The selection of possible constraints is filtered based on the type of the object being edited.

Figure 11 shows a situation in which the user has violated a constraint. The user has attempted to make an *Individual* (*Individual-143*) a subtype of the *Individual*, *George*. But the system author has placed the "Parent of Individual" constraint on the superclass, *Individual*, which does not hold after the subtype operation. The system has displayed an error message explaining the violation in English and in Z (Hayes, 1987). The subtype operation will be refused.

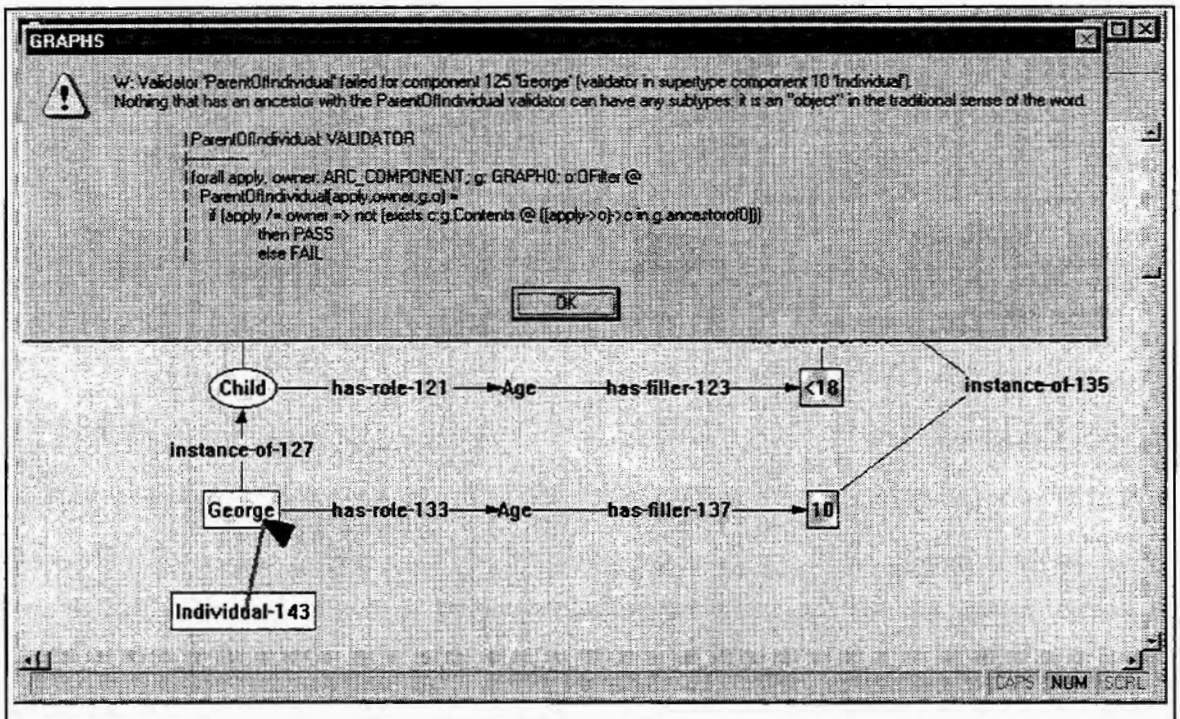

Figure 11: A constrain violation

REFERENCES

Gaines, B. R. (1991). "An Interactive Visual Language for Term Subsumption Languages." *Proceedings of IJCAI-91*, Sydney, Australia, August 24-30, 1991.

Hayes, I. (Ed.) (1987). *Specification Case Studies*. Prentice-Hall, Englewood Cliffs, N.J. 1987.

Kremer, R. (1996a). "Toward a Multi-User, Programmable Web Concept Mapping 'Shell' to Handle Multiple Formalisms." Proceedings of the 10th Annual Knowledge Acquisition Workshop (KAW'96), Banff, Canada. Nov. 6-14, 1996.

Kremer, R. (1996b). "KSI's CMap." http://www.cpsc.ucalgary.ca/~kremer/CMap/index.html. September 30, 1996.

Nelson, M. (1995). *C++ Programmer's Guide to the Standard Template Library*. IDG Books Worldwide, Inc., Foster City, CA. 1995.

Netscape Communications Corporation (1996a). "Developer Information: LiveConnect." Netscape Navigator 3.0 Developer Information, http://home.netscape.com/comprod/products/navigator/version_3.0/developer/mojava.html. September 28, 1996.

Netscape Communications Corporation (1996b). " Netscape JavaScript." http://home.netscape.com/comprod/products/navigator/version_2.0/script/index.html. September 28, 1996.