Chapter 8

Extensions and Variations

The previous chapter discussed the application of Constraint Graphs: languages that can be built on top of Constraint Graphs. This chapter aims at the use of these new languages and the environments in which Constraint Graphs may be used.

The first section discusses the Constraint Graphs extension that translates languages defined in Constraint Graphs to Z specifications. While the ability to translate into Z specifications is interesting, it is not necessarily particularly useful. But this exercise functions as a test case for translating visual languages into more traditional existent textual languages for further processing. For example, the KDraw or Conceptual Graphs language definitions described in the previous chapter could be translated into the native language of their respective engines. A KDraw domain could be translated into CLASSIC to create an expert system. A Conceptual Graphs domain could be processed by PEIRCE (Ellis & Levinson 1992), the Deakin Tool (Lukose 1991), or CGKEE (Munday et al. 1996).

The next two sections discuss two Constraint Graph variants. One variant is a Netscape browser plug-in that allows users to browse Constraint Graphs as active multi-media objects on the Web, modify them, and write them back to the server. The other variant is a multi-user version (which also runs on the Web) that allows multiple users to share the same Constraint Graphs workspace, interacting with the concept maps, and each other, in real time: each of them gets a constantly updated view of the map, reflecting the changes made by the others.

8.1 Translation to Z

The Constraint Graphs program is capable of capturing the *syntax* of other graphical formalisms by inferencing via the type lattice – but the reader should note that it is *not* intended to take on the deeper semantic nuances of any arbitrary graphical formalism. For example, although it can emulate the syntax of KDraw, Constraint Graphs cannot do the inferences of KDraw.

However, the Constraint Graphs program is relatively easy to extend to "compile" a graph into the native notation of a target formalism. As a case in point, it took only a single day to write an extension to output a concept map as a complete Z specification – this automatic generation produced the specifications found in Kremer (1996a; 1996b). The generated Z specification is an extension to the Constraint Graphs Z specification given in chapter 5.

The Z translator writes its output as an HTML file suitable for posting on the Web using the ZSL Z dialect (Jia 1995). The original Z language is full of esoteric mathematical symbols not found in many font sets, so Z is normally not very practical for viewing in non-specialized Web browsers. ZSL is an ingenious pure ASCII version of Z that substitutes very suggestive combinations of ordinary ASCII characters for all of the unusual mathematical symbols. Although it takes a little practice, it does not take a Z-literate person long to become familiar with ZSL. The convenience of ZSL being Web-friendly makes up for the slight loss in mathematical typographical elegance. Furthermore, ZSL is also an input language for Jia's ZTC Z type checker (Jia 1995), which means the Constraint Graphs Z translator output can be type checked by ZTC. An attribute that keeps the developer honest if nothing else.

By a fortuitous combination of syntactic conventions, ZTL can be embedded in HTML documents in such a way that Web browsers can display the document elegantly and ZTL can type check the ZTC code while ignoring the HTML code as comments. This is possible because ZTC ignores all lines that do *not* begin with a tab character. One only has to avoid using tab characters at the beginning of HTML code lines, and place tab characters at the beginning of all ZSL lines. HTML's <PRE> (preformatted) directive is used to enclose passages of (tab-prefixed) ZSL code, which maintains line-breaks in the ZSL code.

The Z translator's output starts by laying down HTML header information, then it includes Constraint Graphs' Z specification:

```
<html>
<head>
<title>A Z Specification for "KDraw4"</title>
</head><body>
<center><h1>KDraw4</h1></center>

This is a Z specification for the KDraw4 visual language definition.  It has
been automatically generated from a constraint graph by CONSTRAINT GRAPHS
(c)1996, Knowledge Science Institute, University of Calgary.  This specification
should typecheck using ZTC (c)1993-1995, Xiaoping Jia, DePaul University.

<pre>
        specification
<a href="graphsz.htm">
        include {graphsz.htm} %</a>
...
```

The Z translator then outputs a section describing the ad-hoc constraints, or *validators* that have been used in the language definition, but are otherwise not part of the Constraint Graphs definition. For example, the KDraw language definition (Section 7.3) uses the ParentOfIndividual constraint (but the Constraint Graphs specification does not), so ParentOfIndividual is listed in the validators section:

```
 ParentOfIndividual: VALIDATOR
--------------
 forall apply, owner: ARC_COMPONENT @ forall g: GRAPH0 @
   ParentOfIndividual(apply,owner,g) =
     if (apply /= owner => not (exists c:g.Contents @
          (apply->c in g.ancestorof0)))
        then PASS
        else FAIL
```

Z requires that all identifiers be declared, so the Z translator must make a second scan through the graph to pick up all new identifiers used in the concept map. There are two classes of identifiers: NAMES (used for components) and ATTRIBUTE_NAMES (used for attributes). Since ZTC is rather particular about identifier syntax (identifiers must begin with an alphabetic character and contain only alphabetic characters, numerals, and underscores) and Constraint Graphs is very liberal about identifier syntax (virtually anything is allowed), the Z translator must make up replacements for unacceptable names and create an identifier translation table. The identifiers section for Section 7.3's KDraw definition is:

```
| Concept, Primitive, Individual, Constraint, Rule,
| Role, mutex, coreferent, has_role, has_constraint,
| if_then, has_filler, has_domain, exception, In_Constraint,
| Includes_Constraint, Min_Max_Constraint, Set_Constraint,
| set_contains, Object_with_Role, instance_of, Proto_Concept : NAME

| borderColor, fillColor, shape, textColor, connector,
| lineColor : ATTRIBUTE_NAME
```

Finally, the Z translator outputs the schema that actually defines the target language. The following schema fragment is a fragment from the KDraw schema showing only the *Concept* and *Primitive* nodes and the *isa* arc. The complete specification is not shown because it's more than five pages long.

```
--KDraw4--------------------------------------------------------------
| FIRST_ORDER_TYPED_GRAPH;
| Concept: NODE_COMPONENT;
| Primitive: NODE_COMPONENT;
| is_a: ISA_COMPONENT;
| ...
|-----------------
| % Node "Concept" restrictions (id=0x8)
| Concept.Level = 3;
| exists x:ATTRIBUTE_VALUE | pos_int~x = 0 @ % (0x0)
|       MakeAttribute(borderColor,x,101,{}) in Concept.Attributes;
| exists x:ATTRIBUTE_VALUE | pos_int~x = 16777215 @ % (0xffffff)
|       MakeAttribute(fillColor,x,100,{}) in Concept.Attributes;
| % shape: seq char = "ellipse" @
| MakeAttribute(shape,attribute_value,101,{CONSTANT}) in
|                                          Concept.Attributes;
| exists x:ATTRIBUTE_VALUE | pos_int~x = 0 @ % (0x0)
|       MakeAttribute(textColor,x,101,{}) in Concept.Attributes;
|
| % Node "Primitive" restrictions (id=0x9)
| Primitive.Level = 3;
| exists x:ATTRIBUTE_VALUE | pos_int~x = 12632256 @ % (0xc0c0c0)
|       MakeAttribute(borderColor,x,101,{CONSTANT}) in
|                                          Primitive.Attributes;
| exists x:ATTRIBUTE_VALUE | pos_int~x = 16777215 @ % (0xffffff)
|       MakeAttribute(fillColor,x,101,{}) in Primitive.Attributes;
| exists x:ATTRIBUTE_VALUE | pos_int~x = 0 @ % (0x0)
```

```
      MakeAttribute(textColor,x,101,{}) in Primitive.Attributes;

...

% Isa "is_a" restrictions (id=0x40)
is_a.Level = 3;
{ Upward, NoDanglingTerminals }
       subseteq is_a.Constraints;
GetTerminal(is_a,1) = Concept;
GetTerminal(is_a,2) = Proto_Concept;
MakeAttribute(connector,attribute_value,100,{}) in is_a.Attributes;

...

% the type hierarchy...
Concept parentof Primitive;
...
```

This schema first declares all the new components, then describes each component, and finally, describes the type hierarchy. Note that the type hierarchy does not seem to be complete – this is because a large part of it is implicitly declared in the other definitions.

The output concludes by adding standard HTML trailer text. The generated HTML file can be displayed as a standard Web document and can be type checked using the ZTC type checker.

Future work on this project includes adding an extension to translate a concept map into the (usually textual) native language of other systems, calling up those other systems for native processing and then translating the results back into a Constraint Graphs concept map. For example, KDraw concept maps could be translated into KDraw's underlying native notation (CLASSIC), a separate KDraw interpreter could be called (possibly elsewhere on the Internet), and the inferences could be written back into the original concept map. Section 9.3.4 expands on this concept.

8.2 Constraint Graphs as a Netscape plug-in

There can be little doubt that World Wide Web (WWW) browsers have caught on with unprecedented success as a popular means of accessing information on the Internet. It makes good sense to take advantage of this success, whatever cause one attributes to it, and use the Web as the base infrastructure for a repository of concept maps. Indeed, concept mapping tools should be cleanly integrated with all the other objects normally found on the Web: text, rich text, still images, animation, and HTML forms, to name just a few of the wide variety of multimedia objects that have become available on the Web.

8.2.1 Concept Maps Embedded in Netscape HTML Documents

The current versions of the Netscape WWW browser allows concept mapping programs, written in C++, to be cleanly integrated with the browser such that users can interact with the concept map as easily as they might interact with a clickable image (Netscape Communications Corporation 1997). Concept maps can be used just as though they were native objects to the browser program. Figure 48 shows a Constraint Graphs concept map embedded in an HTML document. This document and concept map describe part of a C++ class library. The concept map is actually an active instance of the Constraint Graphs program. Therefore, this concept map can be scrolled or re-arranged by simple direct manipulation commands.

All this is possible through the *plugin* Application Program Interface (API) provided by Netscape. Plugins are programs which register themselves with the Netscape browser in association with a specific MIME type (Borenstein & Freed 1993), and which are dynamically loaded by Netscape whenever an object of that MIME type is displayed. Plugins can be either embedded in an HTML document or displayed as a complete document themselves.

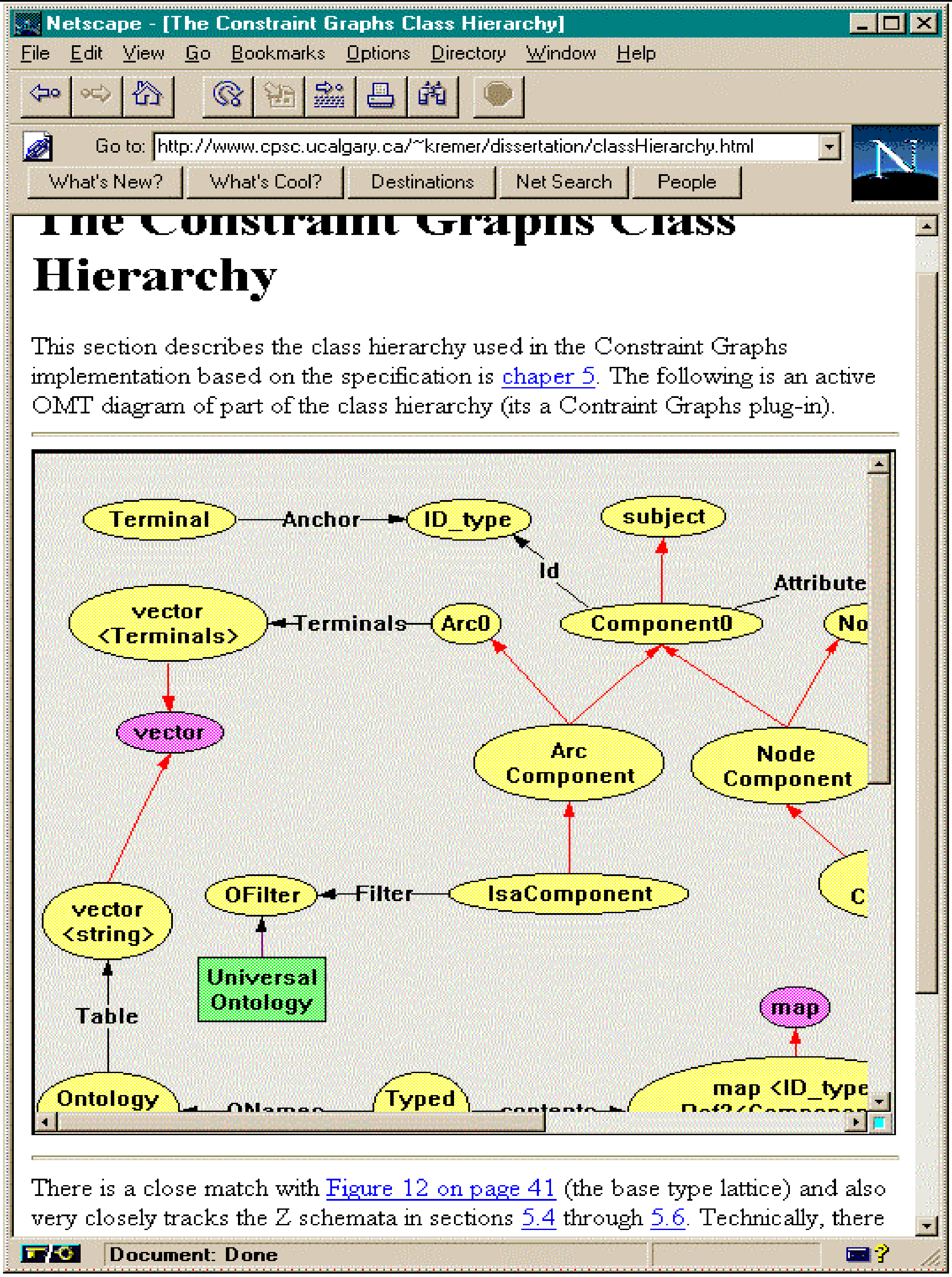

Figure 48: A concept map embedded in an HTML document describing part of a C++ class library.

With the version 3.0 release of the Netscape browser, plugins can masquerade as a Java (Sun Microsystems Inc. 1996) class. This facility is call *LiveConnect* (Netscape Communications Corporation 1996a). Java is an object-oriented programming language tailored for program migration on the Web, and is fast becoming the "native" Web language. In presenting itself as a Java class, a plugin program can make public any internal commands the programmer deems appropriate. Thus, a Java program can assume full control of a plugin program and provide specialized interfaces and operations to a concept map in a way that is convenient for Web programmers.

Furthermore, LiveConnect also provides open communication between Java and JavaScript (Netscape Communications Corporation 1996b). JavaScript is a scripting language that can be written directly into HTML documents, and is even easier to work with than Java. Since concept maps can masquerade as a Java class, and JavaScript can talk to Java, then JavaScript can exert control over concept maps. This allows the user to input elements of Web documents (primarily HTML forms (Grobe 1996)) to act as a large part of the interface to concept maps, giving the end user (or HTML script writer) an unprecedented degree of flexibility and ease of control.

Figure 49 shows a Constraint Graphs concept map embedded in an HTML document. In this example, most of the non-direct manipulation user interface elements that would normally appear as menus or hot keys have been transformed into commands available through a Java class. HTML forms input elements, such as buttons and selection boxes, are scripted with JavaScript to access these commands. For example, the button in Figure 49 labeled "save" is simply defined as:

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

which will call the save() method of the plug-in applet.

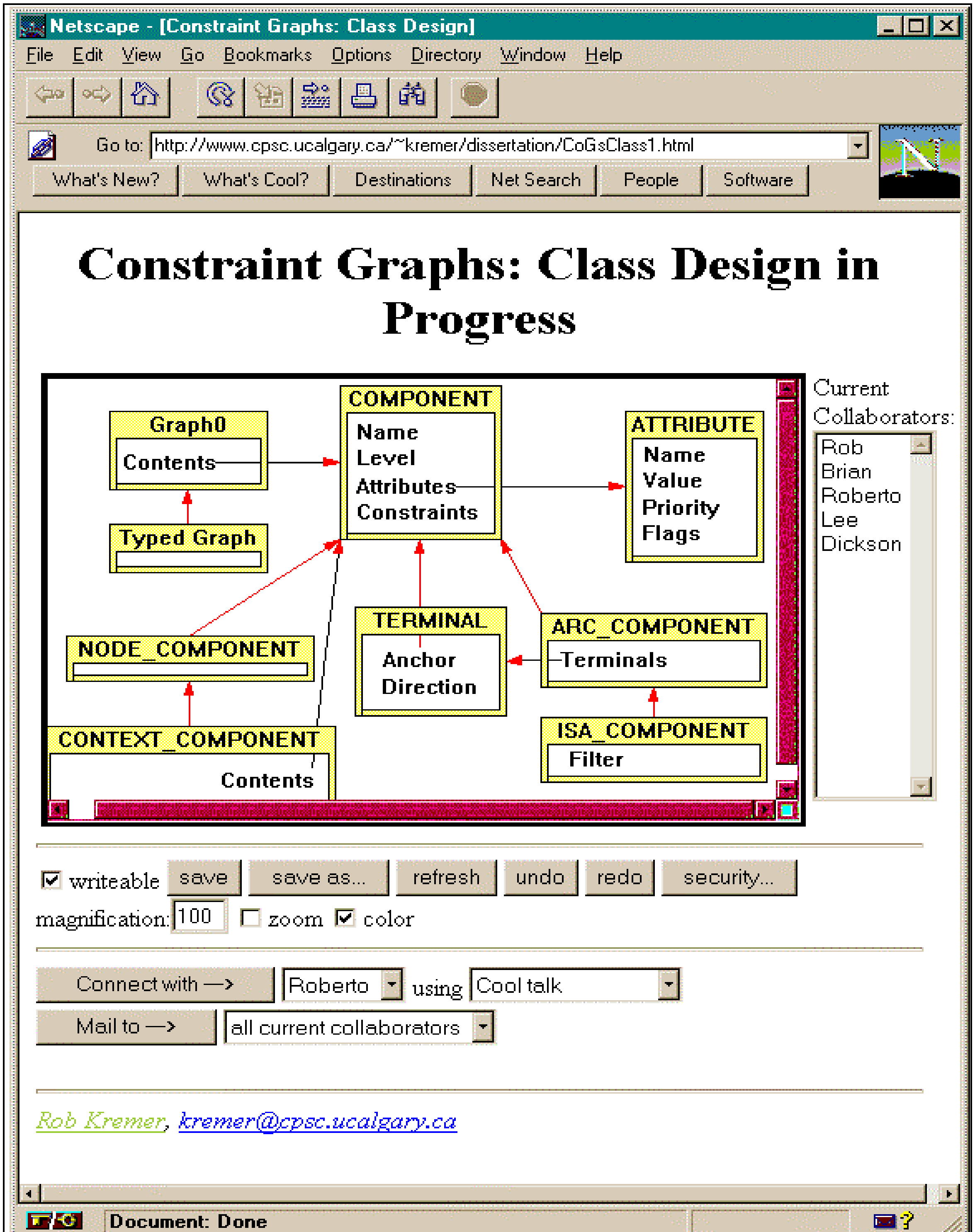

Figure 49: A concept map embedded in an HTML document using HTML forms input elements and JavaScript to achieve a custom user interface.

8.3 Groupware: multi-user, real-time interaction

While concept maps can be very useful as Web documents, their usefulness can be dramatically increased by making them multi-user interactive documents. That is, the traditional distinctions between document author and reader can be dissolved to create a *participant* role, where (sufficiently privileged) individuals can not only browse a concept map but also change the document and store the modified document back to the server. Furthermore, if two or more users happen to access a particular document at the same time, they should be able to smoothly move into a synchronous shared editing session.

8.3.1 Concept Maps Can Be Stored Back to the Server

Allowing users to update concept maps from a Web browser is accomplished through use of the CGI (Common Gateway Interface) available on most WWW servers. A shared concept map is not directly referenced by its normal URL; instead, a custom CGI program is referenced, and this reference is parameterized with a reference to the actual concept map document. When the URL is fetched by the client browser, it is the responsibility of the CGI program to return the concept map data on behalf of the server.

When the client wishes to save the concept map, the same indirect URL is used – but with the concept map data as part of the *post* message. The CGI program is called by the server, and the CGI program copies the data back into the original file on the server.

Of course, this simplistic approach is open to malicious or accidental disruption of the data. A host of security issues come to mind. While these issues have not been addressed so far in this research, there are many well-tested solutions elsewhere. Version control, which can be as simple as maintaining a new version of a file every time it is written back to the server, is one approach; public-key encrypted electronic signatures being verified on a write-back is another.

8.3.2 "Real time", Serendipitous Interaction

Merely allowing users to write concept maps back to the server is not enough. Besides the obvious dangers of users overwriting each others work, users need to be kept aware of other users concurrently accessing the concept map. Users may also wish to find out more about various aspects of the history of the map, such as who else has modified it.

Since shared concept map access is through a server CGI program, this program can take responsibility for recording access as part of the concept map's history, and for coordinating "real time" interaction between concurrent participants. Thus, clients can obtain information about past and current participants from the server (as in Figure 49). Furthermore, since the software architecture uses *command objects* (Gamma et al. 1995) to encapsulate all user updates to a concept map's state, these command objects can be sent to the server where they can be re-broadcast to all concurrent participants. In this way, each of the participants will receive all participants' commands, and all participants' concept maps will eventually arrive at the same state. (There are many finer points to this kind of interaction, but they will not be discussed here.)

This form of interaction has the advantage of being serendipitous; that is, users need not go to any special trouble to set up a "conference": instead, interactive sessions are automatically set up by virtue of concurrent access to the same concept map. (See Roseman & Greenberg (1996) for a discussion of this kind of interaction.)

8.4 Summary

The first section of this chapter discusses the Constraint Graphs extension that translates a Constraint Graphs concept map into a Z specification based on the Z specification for Constraint Graphs itself. This generated Z specification is an HTML document with embedded ZSL (a variant of Z) and can be displayed using any standard Web browser, and it can be used as input (in its entirety) to the ZTC type checker. The Z document serves as an example of simple translation from Constraint Graphs to the native language of other systems whose syntax is captured as a Constraint Graphs definition.

A variant of the Constraint Graphs program can be used on the Web. Constraint Graphs is embedded in an HTML document or is displayed *as* a Web document using the Netscape plugin API. This allows users to work with Constraint Graphs concept maps on the Web just as they might work with them on the stand-alone version of Constraint Graphs. Furthermore, Netscape's recent introduction of LiveConnect allows Constraint Graphs concept maps to interact with other elements, such as HTML forms objects, within an HTML document. This is accomplished by writing the Constraint Graphs plugin to emulate a Java class; as a Java object, the plugin can interact with both the Java environment and with JavaScript.

All of this is a useful concept mapping extension to the Web, but it still conforms to the Web's traditional author/reader roles. It is not difficult to further extend Constraint Graphs, using simple CGI programs to allow real-time, multi-user interaction where each of the users of a particular concept map may be updated with changes made by the other users in real time. This makes for a conference-like shared workspace environment, although a secondary voice channel is often necessary for useful interaction. An advantage of the scenario presented here is that, unlike many groupware tools, this is *serendipitous* interaction, where users do not have to spend undue effort setting up a "conference". Users need only agree to meet at a particular "place" (URL), or may happen to "meet" at a URL by chance.