Aquanet User's Guide

Aerosol Release

4/25/91

Table of Contents

How to use this documentation

If you have never used Aquanet before, Section 1 is recommended reading. It talks about system concepts and defines many of the terms that will be used throughout the documentation. Section 2 is also for new users; it helps you determine whether you can run Aquanet on your workstation or remotely on a server. Then, Section 3 tells you how to start up Aquanet, and briefly discusses some of the mouse and menu conventions the system uses. This section is recommended if you are not currently using X windows or Andrew. It also tells you how to get out of Aquanet.

Sections 4 and 5 will help you use Aquanet. Section 4 is for new users; it steps you through some common use scenarios like browsing a discussion, participating in a discussion, starting a new discussion, and using the schema editor. Section 5 is a command summary for experienced users; it documents Aquanet commands on a menu-by-menu basis. It also talks about the schema editor, the text editor, and key bindings that allow you to enter Aquanet commands from the keyboard.

Section 6 reports known bugs, unimplemented features, and some caveats about system idiosyncracies. Because Aquanet is in the early prototype stage, this section will change frequently. Make sure you have an up-to-date version of this document.

Section 7 is for system maintainers and database administrators. It contains a detailed description of the Aquanet database. You don't need to refer to this section for normal Aquanet use.

List of Figures

1. Introduction

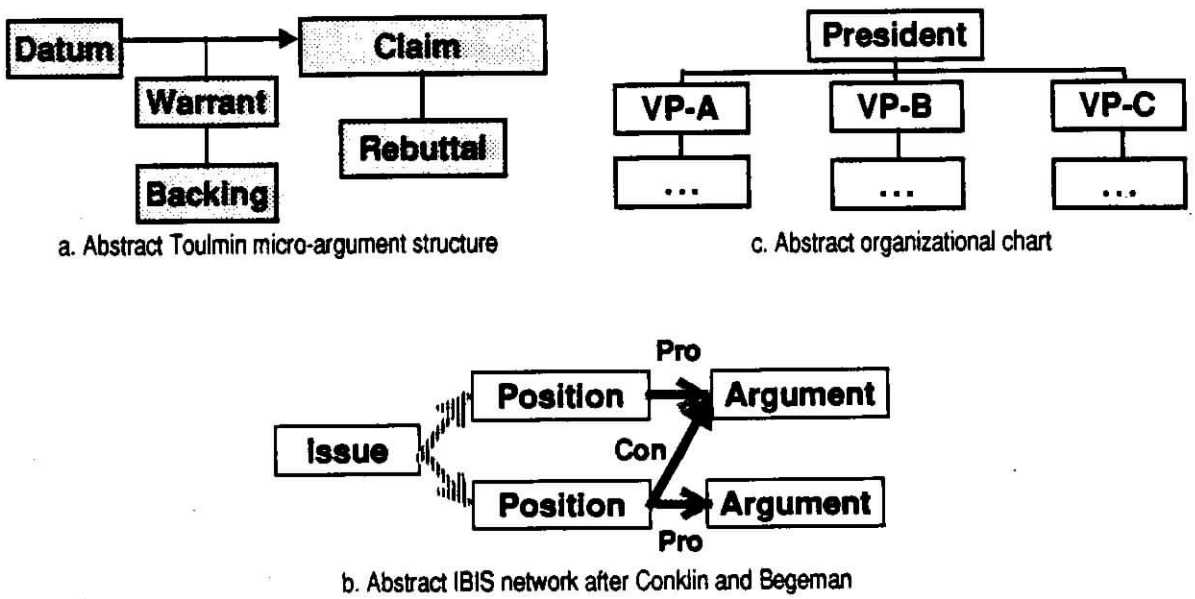
This documentation describes a new system called Aquanet. The remainder of this section will give you a brief overview of the system, give you an idea of how the system can be used, and define some terms used throughout the documentation.

Since Aquanet is a new system, it will change frequently. Make sure you have the latest version of this document. We invite user comments and bug reports - send them to AquanetSupport.parc@xerox.com.

1.1 System overview.

Aquanet is a hypertext tool for people trying to interpret information and organize their ideas, either individually or in groups; we have been calling such activities *knowledge structuring tasks*. Aquanet supports this kind of task by providing facilities for developing, modifying, and using graphically rendered representation schemes to structure information and ideas. Figure 1.1-1 shows three examples of the kind of graphical representation schemes that drove our thinking in developing Aquanet. The tool (in particular, its *schema editor* and *type editor*) allows you to define and change both the graphic appearance and the structure of these representation schemes.

For a more extensive overview of Aquanet, see "Aquanet: a hypertext tool to hold your knowledge in place" (stored as a Framemaker document in /import/aqunet/doc/aquanet-paper.doc and as a Postscript file in /import/aquanet/doc/aquanet-paper.ps).



a. Abstract Toulmin micro-argument structure

c. Abstract organizational chart

b. Abstract IBIS network after Conklin and Begeman

Figure 1.1-1 Examples of graphically rendered representation schemes.

1.2 Why would I use Aquanet.

You can use Aquanet in two different ways. First you can begin a task with the idea that there is a particular methodology that will help you structure your work. For example, designers have found Rittel's Issue-Based Information Systems (IBIS) methodogy helpful in structuring their discussions of design issues, or people doing argument analysis have found Toulmin's theory of micro-argument structure useful in pulling apart complex arguments. Some methodologies, like Toulmin's, may even involve a specific graphical representation scheme (see Figure 1.1-1a). Figure 1.1-1b shows Conklin and Begeman's interpretation of a graphical rendering of Rittel's IBIS methodology.

The second way to use Aquanet is to build up a representation scheme as you work on a task. At the outset of your task, you may start with a very general representation where all your information and ideas are a single kind of thing, and all connections between them are the same kind of connection. Then as you continue your work, you can begin to identify categories of information, and refine the kinds of relationships that exist between them. Aquanet provides facilities for you to extend and redefine your structuring scheme.

1.3 An Aquanet vocabulary.

The Aquanet User's Manual uses some special terms; in this section, we define these terms. To illustrate the terms, we will refer to the following example:

Suppose we would like to discuss whether and how use of Xerox's internal network should be controlled. In particular, we'd like to examine some issues concerning the use of electronic mail. First, we may want to understand an existing argument that concludes that Xerox network administration should eliminate personal and recreational use of distribution lists (dls) and should educate employees about the use of various forms of email, including highly formatted messages.

To do this, we'd like to identify grounds for reaching this conclusion, and perhaps find some rationale that explains why the grounds allow us to reach this conclusion. On examination, we might find that the grounds for this conclusion is a general desire to control Xerox's internet as a corporate resource. We might also infer that the rationale behind this argument is that personal and recreational dls make poor use of a limited resource, thereby rendering its performance inadequate.

This type of argumentation might motivate a general layout to represent the relationship between a conclusion, its grounds, and its rationale. One possible take on this layout shown in Figure 1.3-1. Of course this simple structure does not provide any way of arguing against any part of the argument, or any assertion in general. So, to this general argument representation, we've added another component, a counterargument - a response - that can address either a conclusion, its grounds, or its rationale. This counterargument structure is also shown in Figure 1.3-1..

Using this structure, we can represent our electronic mail argument by mapping the

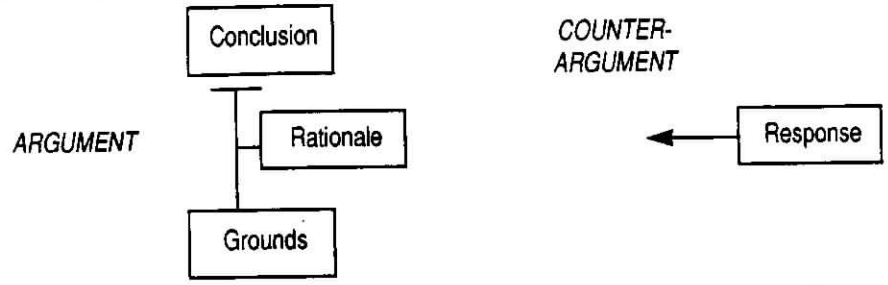


Figure 1.3-1. An example of an argument representation

various assertions we've made into the categories Conclusion, Grounds, and Rationale, as illustrated in Figure 1.3-2. We have added a counterargument which we intend to flesh out later.

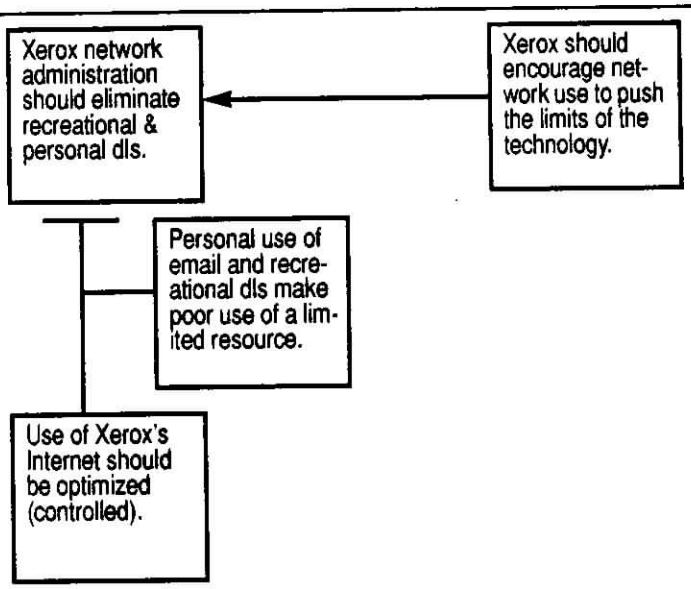


Figure 1.3-2. Applying our simple argument layout to our example.

We will now use this example to define some key ideas and terms in Aquanet.

Knowledge structure.
A knowledge structure is a computer-based representation that depicts objects and their interrelationships in some domain of discourse. Thus, in the example from argumentative discourse presented above, the statements are the objects and their interrelations are defined by our simple description of the parts of an argument.

Graphical knowledge structure.
A graphical knowlege structure is a knowledge structure whose primary presentation is as a graphic display on a computer screen. For the example

shown above, the graphical aspect of the knowledge structure is presented as the spatial layout of the three argument elements and the one counterargument element, and the lines drawn to clarify their interaction. Other examples of graphical knowledge structures are shown in Figure 1.1-1.

Basic object.

Basic objects are the atomic units of Aquanet; they may have properties (such as a name, owner, creation date, time of last edit, and other attributes useful in the context of a discussion or application). In our example, we might decide to define two types of basic objects, a Statement and a Note, where a Statement is an assertion like "Use of Xerox's Internet should be optimized or controlled," "Personal use of email and recreational dls make poor use of a limited resource," or "Xerox network administration should eliminate recreational and personal dls," and a Note is an observation about the schema itself, or the way the schema is being used, "Perhaps we shouldn't be displaying the rationale, since I'm having a hard time coming up with them."

Relation.

Relations express how the basic objects can be connected in larger structures. Standard hypertext links can be thought of as the simplest kind of relation; that is, they are a relation between two objects. In our example, we define a relation called an "Argument," where Statements interact through a relation that has three parts, a Conclusion, Grounds, and a Rationale. We define a second relation, a binary relation, called a "Counterargument," which links a Response with the Statement it responds to.

Node.

Nodes are a general way of referring to basic objects or relations. We use this term to simplify operations that do not distinguish between the two. For example, selection - picking a node from the display by clicking on it with the mouse - is the same kind of operation whether the node is a basic object or a relation since both are represented graphically.

Virtual node.

A virtual node is a display copy of a node. It depicts the same actual node as the original graphic. In the example schema, suppose we would like to use the same statement, "Use of Xerox's internet should be optimized (controlled)," as the grounds for another argument (see Figure 1.3-3) about the use of formatted electronic mail. A copy of the graphic presentation allows it to be re-used in the network (the replicated node graphic is indicated in the figure by thicker boxes).

Role.

In Aquanet, basic objects play roles in relations. In our example, the

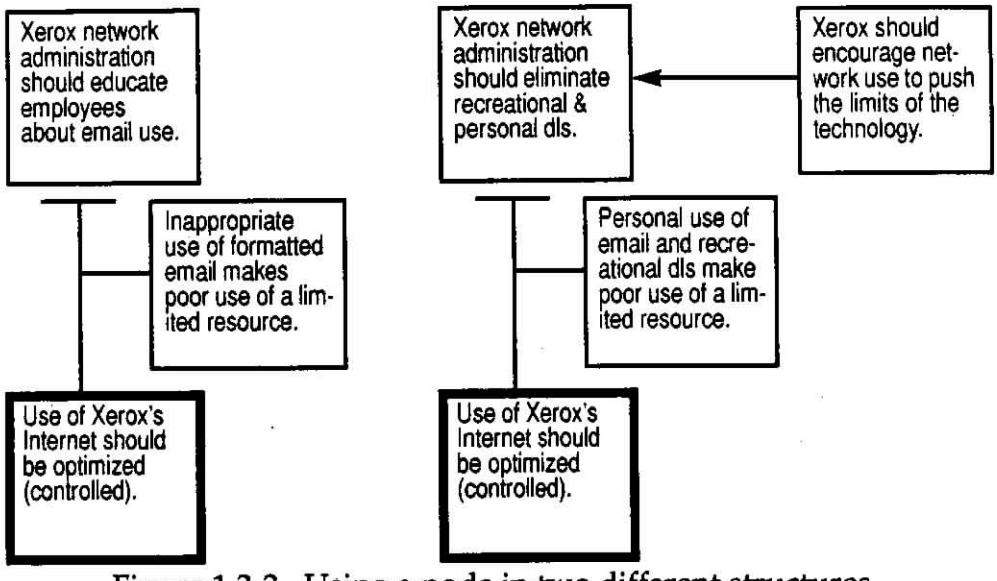


Figure 1.3-3. Using a node in two different structures

statement "Use of Xerox's Internet should be optimized (controlled)" plays the role of Grounds in the Argument relation. The same statement might play another role (for example, Conclusion) in another Argument relation. The notion of roles sharing objects is what allows a user to build up networks; two relations are connected together by the fact that a common object plays a role in each of the relations.

Appearance.

Basic objects and relations both have appearances that specify what they will look like on the screen. For example, Argument and Counterargument relations and Statement and Note basic objects might appear in the forms shown in Figure 1.3-4. Each of the dotted boxes in the two relations indicates a region where a basic object will appear after the role has been filled; the role's name is indicated in the box. The two solid rectangles correspond to the regions of each of the relations. The lines that are defined for each of the relations will appear every time the relation is created.

The appearance of our Argument relation currently places equal emphasis on each of the three roles. Later, we may decide to use the schema editor to change this appearance; perhaps we'd like the person reading the argument to focus on the Grounds and Conclusion of the argument, and de-emphasize the Rationale on the display. Then we might choose to represent the Rationale role with an icon that is small relative to the size of the other two roles.

Because basic objects also have an appearance, precedence will determine whether they take on the appearance of the role they are filling, or whether they will be displayed using their own appearance definition. See the

discussion of node-centric relations later in this section for a more complete description of when the appearance of basic objects takes precedence over the appearance of the relations they are in.

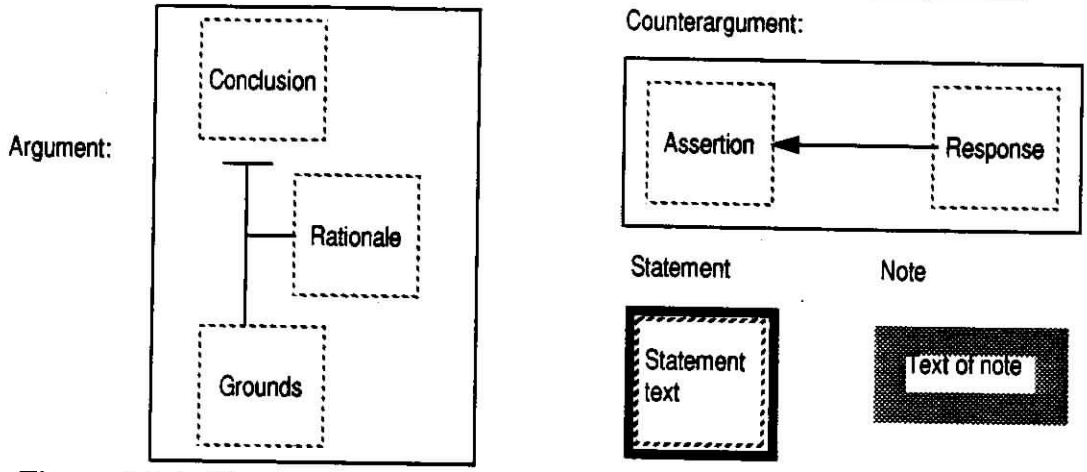


Figure 1.3-4. The appearance of our simple argument and counterargument

It also should be noted that although this documentation is in black and white, appearance characteristics include color as well. The role of color in the design of object and relation appearance will be left to those creating and modifying schemas for individual discussions.

Schema.

An Aquanet schema defines a set of basic object and relation types to be used together. The schema includes a specification of what roles are defined in the relations, and what types of basic objects can fill these roles (see the discussion of slots below). In defining the roles and how they may be filled, a schema specifies how instances of relations can be interconnected. For example, by defining an Argument schema that allows the roles of Grounds and Conclusion to both be filled by an Statement object, the Argument relations can be chained together to form a network such as the one shown in Figure 1.3-5. Likewise, if we allow both Assertions and Responses to be filled by Statements, we can chain Counterarguments to our Arguments.

Aquanet schemas also specify the graphical appearance of each basic object and relation. Aquanet users to interact with schemas through the schema editor described in Section 4.5.

Node-centric and relation-centric relations.

Relations in Aquanet can either be node-centric or relation-centric. This property controls some important aspects of the layout of basic objects and relations. In a relation-centric relation, the nodes filling entity-valued slots are reshaped to fit the slots, and the slot position is as defined by the

relation. On the other hand, in a node-centric relation, a node filling an entity-valued slot stretches the relation to conform to its position, and any node retains the size defined for its type, even if it is filling a relation's slot.

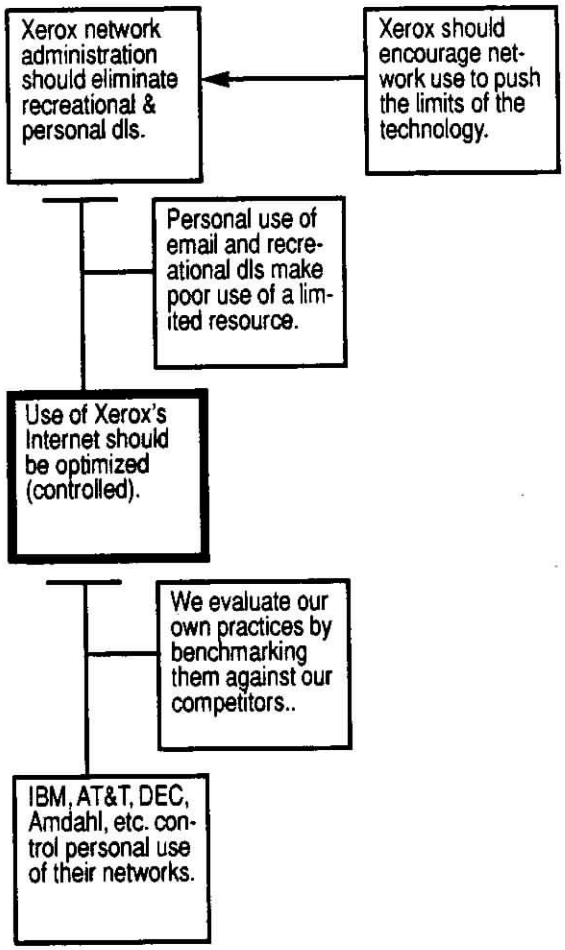


Figure 1.3-5. Interconnection though shared elements.

Slot.

Slots are the way a role or another kind of attribute is associated with a specific value. In our electronic mail example, one slot might called the Grounds slot, and for a specific Argument relation, it might be filled with the statement "Use of Xerox's Internet should be optimized (controlled)."

In general that slot might be constrained to accept only values that are statements. This kind of slot, slots that take basic objects or other relations as values, are referred to as e-slots. Only relations can have e-slots.

A second kind of slot associates a "primitive value" with the slot. By primitive value, we mean a string or number as opposed to an object. These we refer to as p-slots. In the electronic mail example, we might want to

associate a slot called "says-who" with a statement, and specify that it be filled by a string. This way, if the person making the argument wanted to supply an authority she was citing, she could fill in the slot with text. Both basic objects and relations can have p-slots.

Multiple valued slots.

Multiple valued slots are e-slots that can be filled with more than one node. For example, we may want our Argument relation to allow a Conclusion to be support with more that one Grounds; thus we'd be able to specify a number of Statements that fill the Grounds role.

Discussion.

Schemas are used in the context of a discussion. A discussion may be a multi-user conversation using a schema to structure its content, or it may simply be a single user's network she is constructing based on a schema. Discussions may be restricted if the participants want to limit access to members of their own group.wide participation.

2. System Requirements

Aquanet can be used in two ways. First, if you are running on hardware that meets the system hardware requirements, you can run the Aquanet program on your own processor and display the Aquanet window on your screen. Second, if you are running on a system that can run X windows, you can run Aquanet from a server and display the Aquanet window on your screen. Both modes of use are covered in this section.

2.1 Running Aquanet on your own processor.

Aquanet runs on any Sun 3 or 4 workstation, including SPARCstations. A color monitor is best for displaying the Aquanet window, but a monochrome monitor may also be used (with some potential cost in being able to distinguish different node types in the network display). Aquanet can only be run from machines on the network.

Running Aquanet on the Liveboard.
You can run Aquanet on the Liveboard for meetings where one person is "driving," while the others are contributing verbally to the construction of the network. Since the Liveboard is a monochrome display, the caveats discussed above apply.

2.2 Running Aquanet from an Aquanet server.

You can use Aquanet from any system that can display X windows, including Mac IIs (and, in the future, Symbolics machines). Your terminal will display the Aquanet window and a designated Sun will act as an Aquanet server. A monochrome display will work, although Aquanet uses color to help you distinguish among the node types it displays.

2.3 Other helpful documentation.

Supplementary documentation - like Unix man pages or the Andrew Help System - might be useful, although we do not expect it to be necessary. For example, if you were writing a report that used text from Aquanet, you might want to read the documentation on Andrew's text editor, EZ.

3. Starting up Aquanet

This section covers the basics of getting in and out of Aquanet. An Aquanet startup script will set up your system to run Aquanet according to the hardware configuration you describe (color or monochrome display, Mac or Sun, and other scripts that are set up as other X window implementations are available).

3.1 Starting Aquanet.

The first time you run Aquanet, you will need to execute a bringover script. This script installs the necessary files on your home directory. To do this, type

bringover aquanet

at the prompt (often a "%"). If you already have the necessary files on your home directory, the bringover script will replace them (and rename the old files by appending their dates to their names).

You will also need to insert the following line in your .login file:

source /import/aquanet/top/enable

If you don't choose to add this command to your .login file, you will need to type it into the shell window each time you start Aquanet. If you run aquanet before you login again (or before you source your .login), you will also need to type it in.

Starting Aquanet from a Sun running X.
From a shell window on the display where you want the Aquanet window to appear, type

aquanet

at the prompt (often a "%"). The outline of a window will appear. Clicking with the left mouse button anchors the window and causes it to display itself at the designated position.

If you want to run Aquanet from a remote server and have the Aquanet window appear on your display, type

aquanet [*-display <yourhost>:<displaynumber>*]

where *<yourhost>* is the name of the display's host machine (to find out your display's host name, just type *hostname* at the prompt). If you have a single display, *<displaynumber>* will be *0.0*; a two-headed display may require a different argument (for example, *0.1* to refer to the second display).

Starting Aquanet from a Sun not running X.
If you are not currently running X windows and Andrew, you will need to

bringover and enable each of them. If you are unfamiliar with the procedures for doing this, you may need to consult your user support person. After you've done this, to start X windows from a Unix login shell, type

runx twm -- xnews

at the prompt. This command will start X windows (using the twm window manager and the Sun Merged NeWS/X server). When this has completed, you will have a shell window from which you can start Aquanet by simply typing

aquanet

at the prompt (often a "%"). The outline of the Aquanet window will appear on your display. Clicking with the left mouse button anchors the Aquanet window and causes it to display itself at the designated position.

Starting Aquanet from a Mac *(not available yet).*
To start Aquanet from a Mac running the standard MacOS, retrieve the Aquanet application using Tops. It will be on the Aquanet directory. Double-clicking on its icon will start Mac X. You will not actually be running Aquanet on your Mac, but rather will be running Aquanet on the Aquanet server and displaying it on your Mac.

Using your .aquanetrc file to change important variables.
Aquanet uses a file called .aquanetrc on your home directory to set variables that dictate system behavior. The bringover script that you used to set up Aquanet installed an initial version of this file. You can edit .aquanetrc to change the values of these system variables.

The first of these variables is aquanet.PollingInterval; it tells Aquanet how frequently to go to the database for updates. Currently, aquanet.PollingInterval has a default value of 30 (meaning 30 seconds); if you are the only user, you may want to raise this value to get better performance. In a multi-user highly synchronous situation, you may find 30 seconds to be too long between updates; to correct this, you can lower the value (eg. aquanet.PollingInterval:15).

If you prefer a different initial Aquanet window size, modify aquanet.DefaultSize (e.g. aquanet.DefaultSize:1100x800).

Currently, you must check out any node you plan to edit; if the value of the variable aquanet.CheckoutOnSelect is true, any node you select will be automatically checked out to you (so you'd set aquanet.CheckoutOnSelect:true).

Finally, if you want Aquanet to open the same discussion each time you start the system, you can set the variable aquanet.DefaultDiscussion (eg. aquanet.DefaultDiscussion:demo).

3.2 *Mouse and menu conventions.*

At the present, Aquanet inherits many of its mouse and menu conventions from Andrew and X windows. This section discusses these conventions.

Menus *ala* Andrew.
Andrew has pop-up menus that stack - that is, the submenus (sometimes referred to as menu cards in other documentation) group together sets of commands. The submenus stack from left to right; if you move the cursor to left, the menu whose edge you've touched will be revealed. To choose a menu option, keep the mouse button pressed while moving the cursor to the menu item you want. When the menu item reverses (white text on black), release the button to choose it. Figure 3.2-1 shows you an example of an Andrew-style menu used for Aquanet.

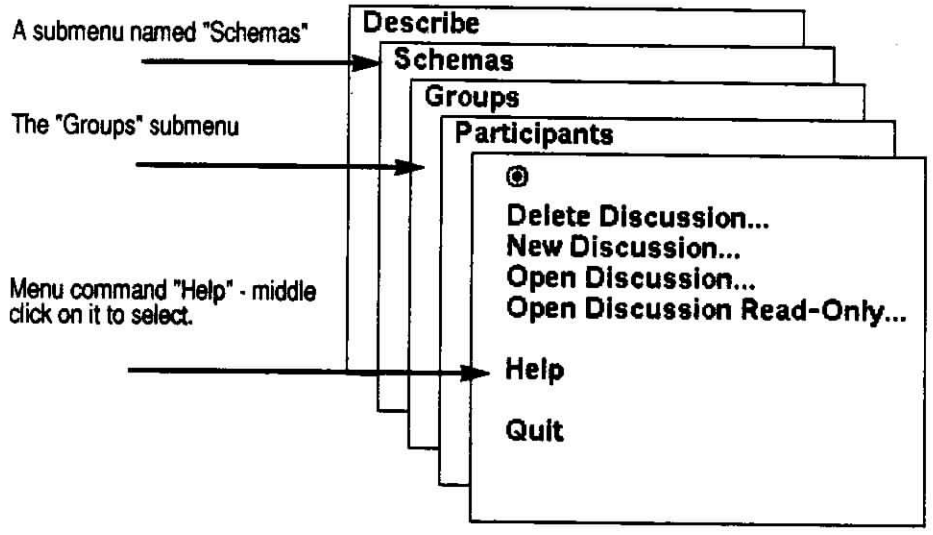


Figure 3.2-1. An Andrew/Aquanet menu with five submenus

Mouse button conventions.
In the Aquanet window, you use the left mouse button for selection. For example, in the Aquanet network display, clicking with the left mouse button in a node's area should cause the system to highlight the selected node and all copies of the node. Left clicking will also activate the other panes of the Aquanet window and select items from them. However, in an overlapping display (say of one node partially obscuring another), left clicking will not bring the node to the top; explicit commands are provided for that purpose. In text applications, right clicking extends the selection you have made with the left button.

Inside a window, the middle button is reserved for application-specific commands; thus all Aquanet commands are on the middle button pop-up menu. These commands change with context - only the appropriate commands appear on the menu. So you will see different items on the middle button menu depending on the situation.

Window commands may be either on the middle mouse button or on the right mouse button, depending on how your Aquanet system is set up; to get them, you must click in the window's title bar. You can shrink windows to an icon (and expand them to full size again), bring the window to the front if it is partially obscured, put it in the background if it is obscuring another window, redisplay it, move it, shape it, and close it - in short, you can perform all the standard kinds of window operations. If you select "Kill Window," you will terminate your Aquanet session.

If you just want to move a window, holding down the right (or middle) mouse button inside the window's title bar will give you a wire frame corresponding to the size of the window. You can use this frame to drag the window wherever you want it on the screen. Holding down the right mouse button in the upper right hand corner of the window allows you to change the size of the window.

Scrollbars.
Aquanet scrollbars are on the left side of the window panes; each pane may have its own separate scrollbar. Clicking the left button in the s scrolls down, the right scrolls up. The whole scrollbar is taken to represent the length of the contents of the window, the white part represents the part of the text or graphics that is shown on the screen, and the rectangle inside that represents the relative length of the current selection.

Dialog boxes.
Sometimes Aquanet will use a dialog box to present you with a series of alternatives. Figure 3.2-2 shows you a typical dialog box. The cursor will turn to a large dot when you are selecting an item from it; the darkened option is the default. You can also select an item by typing its first letter and hitting a carriage return.

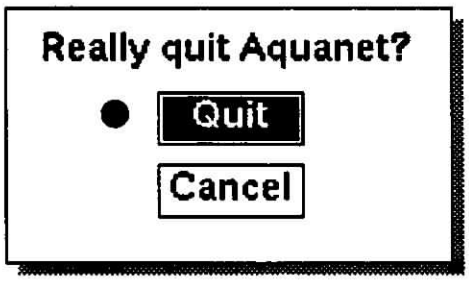


Figure 3.2-2. An Andrew/Aquanet dialog box

3.3 Quitting Aquanet.

Quitting Aquanet is simple, and regardless of how you do it, the discussion network is left in a consistent state.

How your work is saved.
Aquanet saves your work as you go; that is, whenever you complete a change, either to the network or to text, your work is written to the Aquanet database. (When you are editing text, your changes are written when you either use the Check In command, or when you select another node or activate another part of the window.) This ensures consistency in the event of a system crash. So, when you quit Aquanet, you don't have to go through any particular motions to save your work.

Quitting Aquanet.
Clicking the middle mouse button in the Aquanet window will give you a stack of the current submenus. On the top one, you will see a Quit option. Quit terminates your connection to Aquanet and closes the Aquanet window. Aquanet will prompt you (via a dialog box like the one shown in Figure 3.2-2) to either confirm the Quit or Cancel.

If you just want to change discussions, you don't need to Quit and restart. Instead you can choose Close Discussion from the Discussion options (available on the middle mouse button menu in the main view whenever a discussion is open), and then open a new one.

Returning to the state you started in.
Because Aquanet is a multi-user system, there is no "abort session" command that returns the discussion network to the state it was in when you started. If you want to remove your changes, you must go through and delete them individually. This way, if someone else is participating in the discussion at the same time you are, you will be aware of their subsequent changes to objects or relations you have created.

4. Using Aquanet

This section assumes that you've successfully started up Aquanet, and have its window on your display. First we will describe the Aquanet window. The subsections following that will walk you through how to participate in an existing discussion, how to start a new discussion, how to use the schema editor to create and modify schemas, and how to use Aquanet as a single-user tool.

4.1 The Aquanet window.

Figure 4.1-1 shows the Aquanet window.

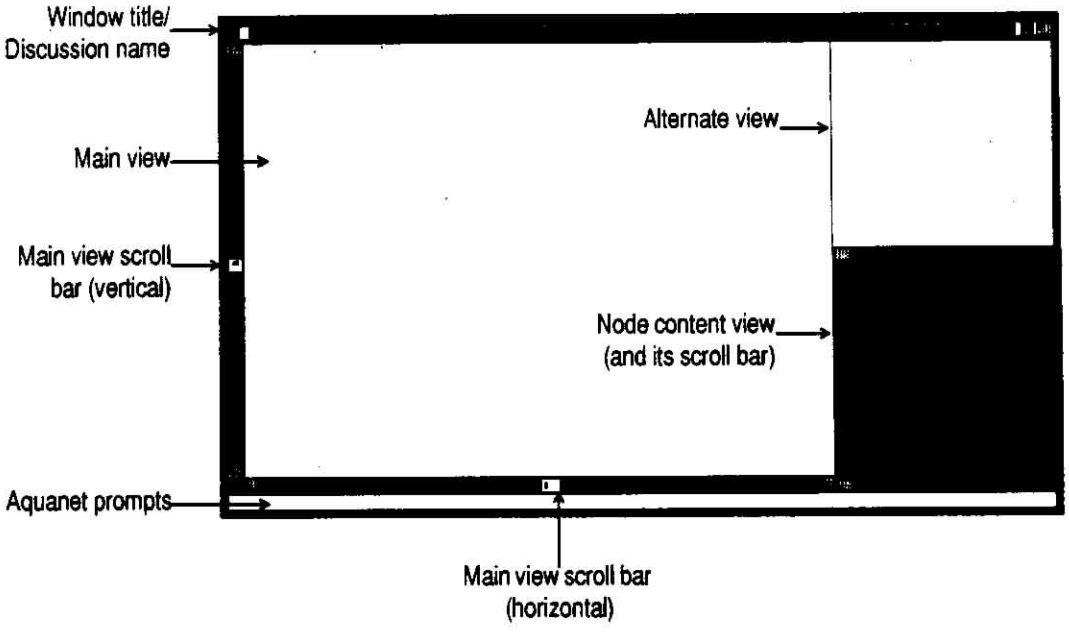


Figure 4.1-1. The Aquanet window

The Aquanet window has three panes - a main view (the big region on the left), an Alternate view (the upper right pane), and a Node content view (the pane on the lower left). Each of these panes may have its own separate scroll bar so you can view contents that are not currently displayed in the pane. The window has a title bar that will change to reflect the name of the current discussion. Aquanet will sometimes prompt you for input; these prompts may be displayed either in the strip running the length of the bottom of the window, or in a separate dialog box.

The Main view.
The main view displays a portion of the discussion (the rest should be visible if you scroll). You will be able to see a graphic representation of the instances of basic objects and relations that have been created during the course of a discussion. From this pane you can select any node (a basic object or relation), and see its contents in the Node content view. You can also move a nodes presented in this

display by pressing down the left mouse button and dragging it to a new position; attached nodes may be dragged along with the node you are moving.

The Alternate view.
The Alternate view is used for a variety of purposes. Sometimes it is to present you with a selectable list of options. For example, if the Aquanet prompt asks you for a name of a discussion to open, the list of existing discussions will be presented in the Alternate view pane. In general, it is wise to check this pane for options when you are prompted to enter the name of a discussion or group. You can also use it to look at a filtered list of all the nodes in the discussion.

The Node content view.
When you have selected a node in the main view, its contents are displayed in the Node content view. These contents are usually editable (for example, you can change the Name field, along with other primitive slots in the node). If the pane is gray, it means that no node has been selected.

4.2 Aquanet and WYSIWID

Aquanet is intended to be used semi-synchronously like an on-line bulletin board; other users may be accessing the same Aquanet discussion at the same time that you are, but your session only polls the database periodically (at a default interval of thirty seconds). Aquanet is thus a WYSIWID system - What You See Is What I Did. You will see changes soon after another user is done making them.

Aquanet's locking mechanism prevents users from changing the same node at the same time (although several users can get into a tug-of-war trying to move portions of the network; see the warning in Section 6). The system signals potential conflicts by warning you that the node you are selecting is locked. Aquanet will tell you which user has the node locked, so other communication (a phone call, a loud shout) can be initiated if you want to change the same node.

When you select Edit on a node, or are creating or deleting a node, you initiate a lock on that node that prevents someone else from inadvertantly editing the same node. When you release a node by either selecting some other node, deselecting all nodes by clicking in the main view, or by checking the node back in, other users can edit the node.

Because Aquanet only polls the discussion periodically (as specified in your .aquanetrc file by the variable aquanet.PollingInterval), if you suspect someone else has changed a node, and you aren't seeing the latest version, you can force an update by selecting Update from the discussion menu (Figure 4.3-3 shows you the discussion menu).

4.3 Participating in a discussion.

Aquanet is a group discussion tool; this section talks about how you can participate in such a discussion, or just browse existing discussions.

Opening a discussion.
To open a discussion, press down the middle mouse button while the cursor is in the Aquanet window. A menu will appear like the one shown below. With the middle button still down, select Open Discussion as shown in Figure 4.3-1. If you are planning on only browsing the discussion rather than contributing to it, you can choose to Open Discussion Read-Only.

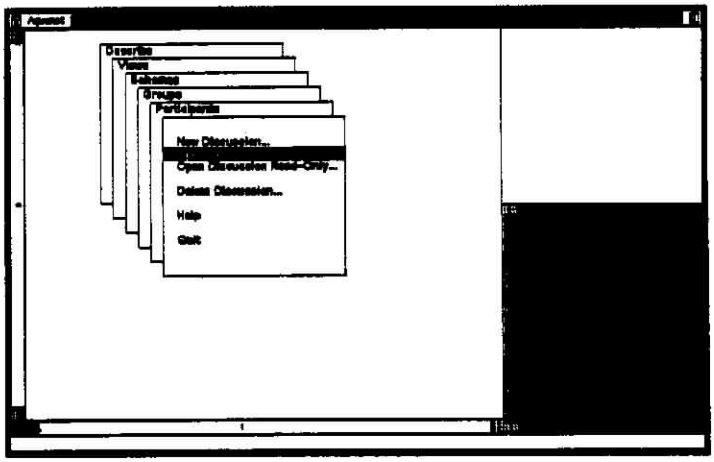


Figure 4.3-1. Opening a Discussion

After you have chosen one of the two Open Discussion commands, a list of all current discussions that you have access to will appear in the Alternate view as shown in Figure 4.3-2. Choose a discussion by left clicking on an entry in the list. Access to discussions is sometimes limited by groups (i.e. you must be a member of a particular group to participate in a "restricted" discussion), but other discussions are unrestricted - anyone who has access to Aquanet can participate.

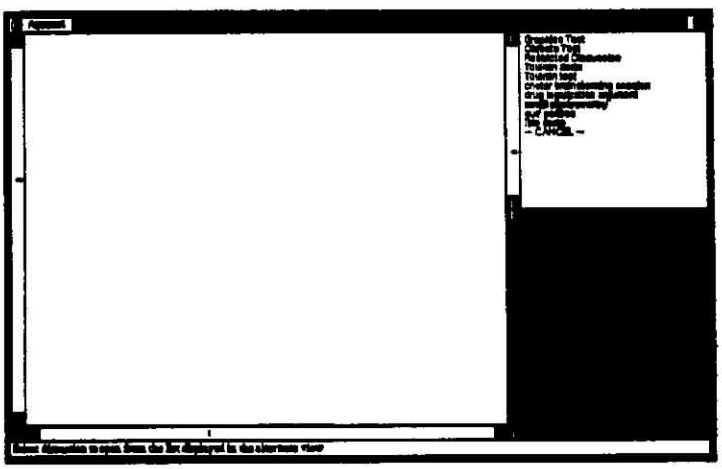


Figure 4.3-2. Selecting the discussion you want to see

Once you have selected a discussion, some portion of it will appear in the main view. The discussion is organized according to a schema, a graphical knowledge structure that was chosen when the discussion started and that can be modified during the course of the discussion as needed. Section 4.5 goes into detail about how you can create and modify schemas. Some aspects of the schema - such as any defined regular spatial layouts - will be obvious from your initial view of the network.

Browsing a discussion network.
After the main view of the discussion has been displayed as in the example shown in Figure 4.3-3, you can begin to browse it. You can use the vertical and horizontal scroll bars on the main view to shift focus on various parts of the network, or you can use the discussion display options that appear on the stack of menus that will popup when you press the middle mouse button (also shown in Figure 4.3-3). Zoom (not currently implemented) broadens your field of view, Pan-Out (not currently implemented) narrows it, and Update goes out to the database to make sure you have the most current version of the network (most useful if you suspect someone's changing the discussion while you're reading it).

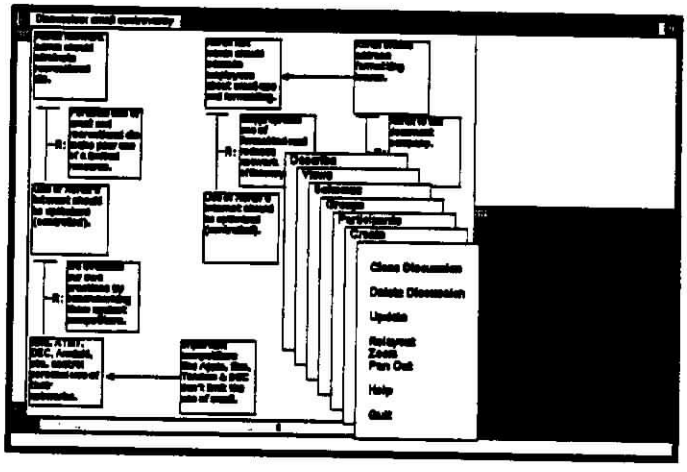


Figure 4.3-3. A discussion and the discussion menu

You can also investigate the contents of individual nodes by selecting them with the left mouse button and reading the their contents in the Node content view (see the example in Figure 4.3-4). When you select a node, all copies of that node are highlighted; this feature is useful for finding nodes that fill slots in several relations. Note that this is true in Figure 4.3-4; the two copies of the same node are both highlighted.

Two other important operations for browsing are Top and Bottom; they appear on the Node submenu (the top untitled submenu) when you have either a relation or basic object selected and you press the middle mouse button. Since both relations and nodes can be stacked on top of one another, Top and Bottom are useful for

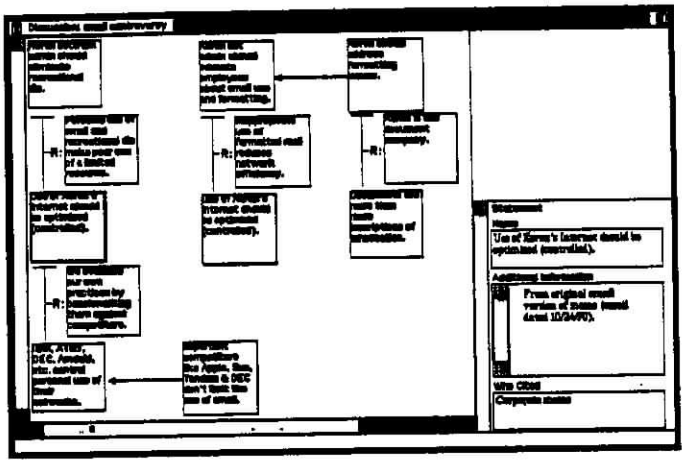


Figure 4.3-4. Selecting a node and looking at its contents

revealing nodes that are hidden. In particular, if a node is partially hidden from view, you might want to select it (by clicking with the left button) and choose Top. Or if you suspect a node is completely obscuring another node (or nodes), successive applications of the Bottom command will help you leaf through the hidden nodes.

Participating in a discussion.
If you have opened a discussion without using the read-only option, you can contribute to the discussion in several ways. First, you can change the way the discussion is laid out. Second, you can add to or modify its structure by adding basic objects or relations, or by using existing basic objects in different roles in the network by putting them in other relations. Finally, you can edit the content of a node - you can change the text or values displayed in the Node content view. You can also change the schema underlying the discussion and alter the structure in fundamental ways, but that will not be addressed until Section 4.5 on the schema editor.

Changing the layout.
Any selected node can be dragged around the main view using the left mouse button. If the node is part of a larger network with a fixed layout, the network will also move with node.

Warning: When you leave a discussion, only the x-y position of nodes is saved; if you have made changes in the Top/Bottom orientation of nodes, they may be lost.

Creating new basic objects and relations.
One way to contribute to a discussion is by creating new basic objects and relations. To create a new object (or a new relation), left click in the background of the network to de-select any previously selected nodes; the Node content view will become gray if you have nothing selected. Then use

the middle mouse button to bring up the set of discussion menus. Pull over to the Create submenu, and you will see a list of the possible objects and relations you can create.

Figure 4.3-5 shows the Create submenu that would appear in a discussion that uses the Simple Arg schema. It has two kinds of basic objects, a Statement and a Note (intended to be something like a post-it), and two kinds of relations, an Argument and a CounterArg. In the figure, a Statement is being created. The graphic representation of a Statement will appear where you last left-clicked in the main view.

If you create a new basic object or relation, and can't see it, it is possible that last place you left-clicked is not currently visible in the main view. Or if you have never left-clicked in the main view, the new node has been placed in the upper left corner of the main view. You may have to scroll to find it.

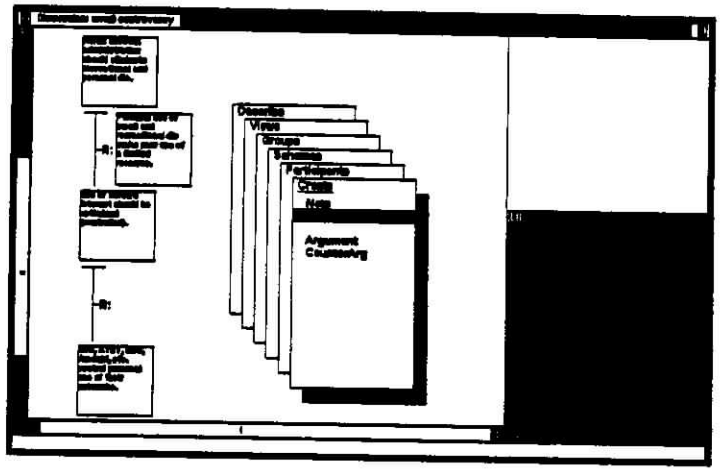


Figure 4.3-5. Creating a basic object

Once you have created a new basic object or relation, you will probably want to edit its content. To do this, left click on the new node, and its content (empty) will appear in the Node content view. In Figure 4.3-6, the new Statement created in Figure 4.3-5 has been selected. Note that the Statement is highlighted; if there had been any other copies of the Statement (e.g. if it had been used in one or more relations), they would be highlighted as well.

Editing the content of a node.

To edit the content of a node, first click in the Node content view (at the lower right of the Aquanet window), then use the middle button to bring up the editing menu shown in Figure 4.3-7. Selecting Check Out from this menu *locks* the node; that is, while you're changing the node contents, no one else can be changing them.

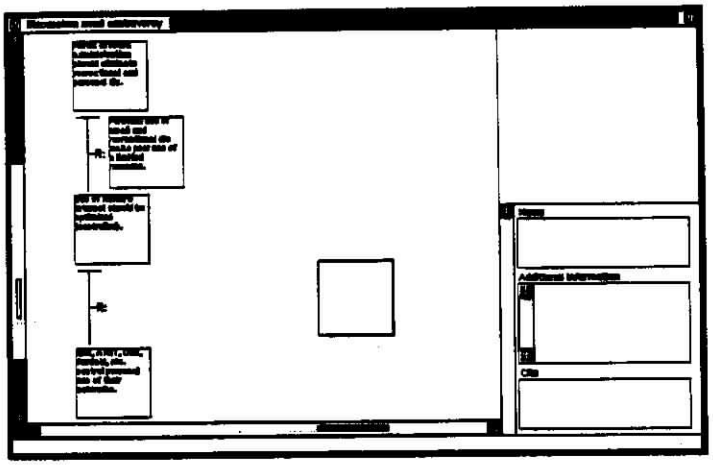


Figure 4.3-6. Selecting the new basic object

The text editor uses the familiar copy-cut-paste model. Typing will go where the caret appears, and the current selection is highlighted in reverse-video. There are other commands on the text editing menu, but they are mostly unneccessary in this context.

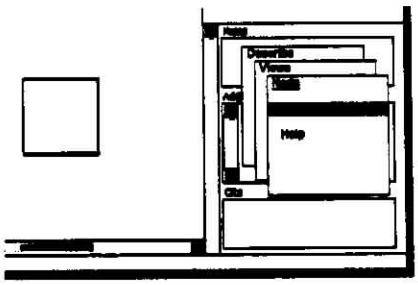


Figure 4.3-7. Editing the content of a basic object

After you've finished the edit, click anywhere in the main view to unlock the node. This way other discussion participants once again have write access to the node and will see the current version of its contents. Any part of the node's contents that have been changed (in many cases, the Name p-slot) will be updated in the main view. You can also explicitly check the node back in by choosing the Check In option in the Node submenu. Figure 4.3-8 gives you an example of the results. You can also choose options on the Node submenu to save or reset the node's contents without checking it back in.

Filling a relation's slots.

Another way to participate in a discussion is by adding structure - putting basic objects in relations. To do this, select the relation you want to add to. The middle button menu will then include a submenu called Slots. From

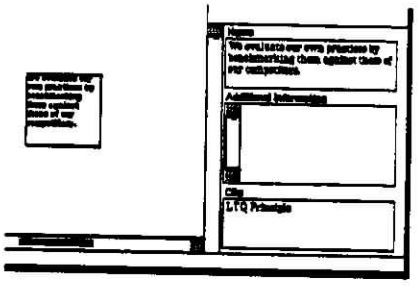


Figure 4.3-8. Completing the edit

that menu, select Fill (Fill with New will create a new basic object of the appropriate type in the slot). Figure 4.3-8 shows the Slots submenu.

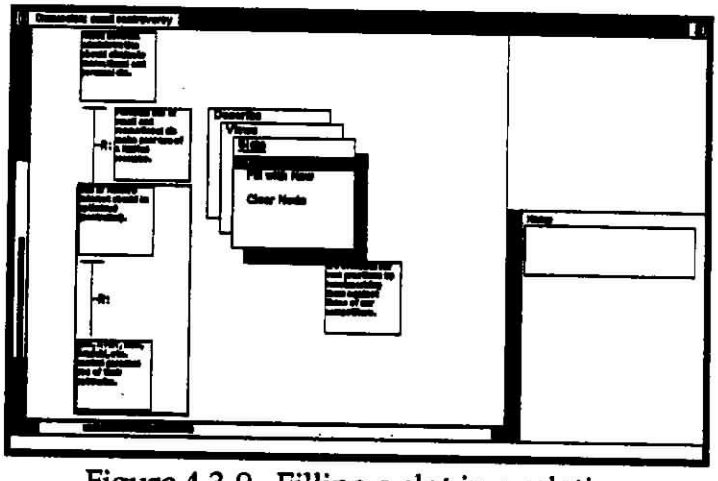


Figure 4.3-9. Filling a slot in a relation

Fill will ask you (by way of the dialog box shown in Figure 4.3-10) to click on the name of the slot you want to fill in the selected relation. In this case, we've selected the Argument's Rationale slot.

Fill then asks you to specify which basic object to put in the slot. In this case, we've clicked on the Statement created in Figure 4.3-6. Figure 4.3-11 shows the results of this operation.

Creating interconnected relations.
Still another way of participating in a discussion is to build up networks of interconnected structure. In Aquanet, networks are built up through relations sharing elements. In the example we have been developing, suppose we want to make a counterargument to the statement, "IBM, AT&T, DEC, Amdahl, etc. control personal use of their networks."

We can do this by making the statement a False-Claim in a CounterArg relation. To do this, we select the Statement we want to share (by left

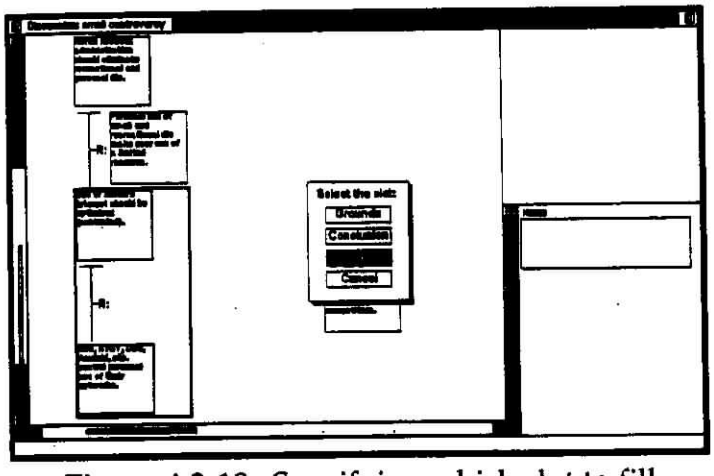


Figure 4.3-10. Specifying which slot to fill

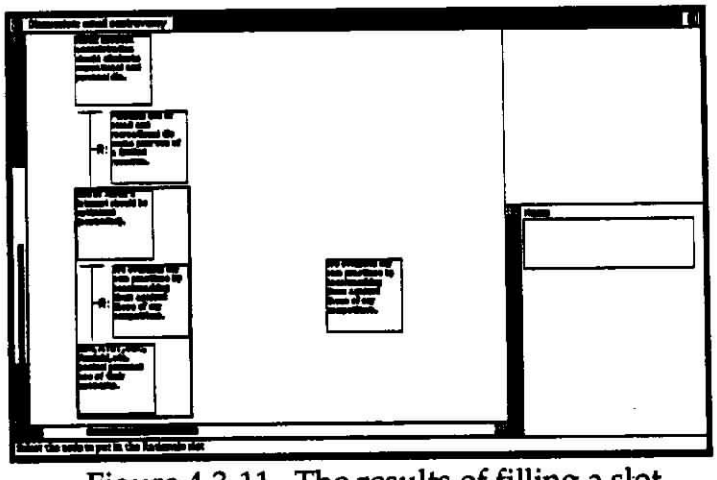


Figure 4.3-11. The results of filling a slot

clicking on it). Pressing the middle mouse button down, and pulling over to the Expand As submenu gives us the options shown in Figure 4.3-12. Note that we can use the Statement in a variety of roles in the new relation that will be created as a result of this operation.

Figure 4.3-13 shows the results of the Expand As operation. A new CounterArg relation has been created.

Deleting basic objects and relations and clearing slots.
If you want to delete any of the basic objects or relations you have created, select the basic object or relation you want to delete and use the Delete option on the Node submenu.

Only the copy you have selected will be deleted. You can only delete basic objects you've created. It is also possible to remove a basic object from a slot

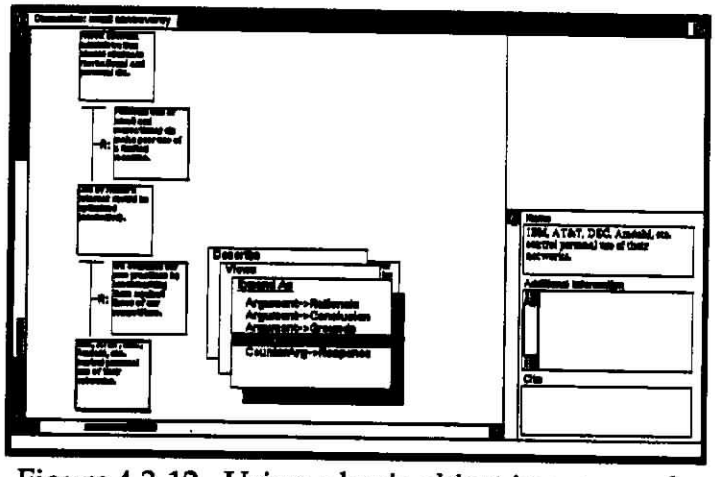


Figure 4.3-12. Using a basic object in a new role

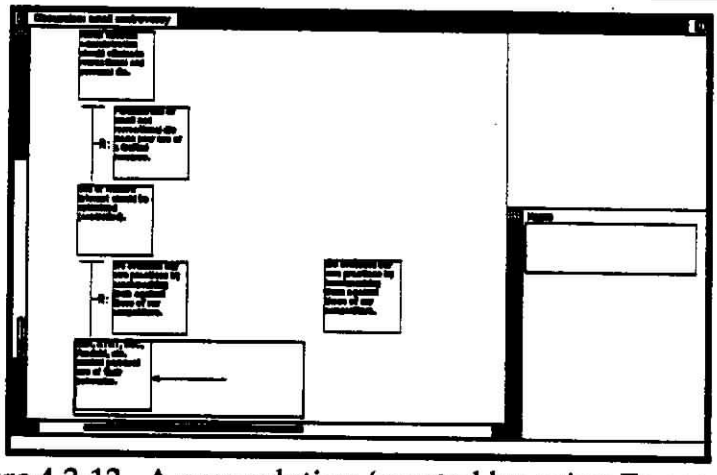


Figure 4.3-13. A new relation (created by using Expand as)

without deleting it by using the Clear Node option. To do this, select the relation containing the slot in question, and use the Clear Node option on the Slots submenu. Clear Node will prompt you to select the slot you want to clear. The basic object will still be in the same position after you've removed it from the slot, but it will no longer be in the slot. Hence you can drag it to some other convenient (and less confusing) place in the main view.

For further examples of creating and using objects and relations, see the next section.

4.4 Starting a new discussion.

This section describes how you can start a new discussion. With your cursor in the main view, press the middle button. You will see the menu shown in Figure 4.4-1. Keeping the

mouse button depressed, select New Discussion.

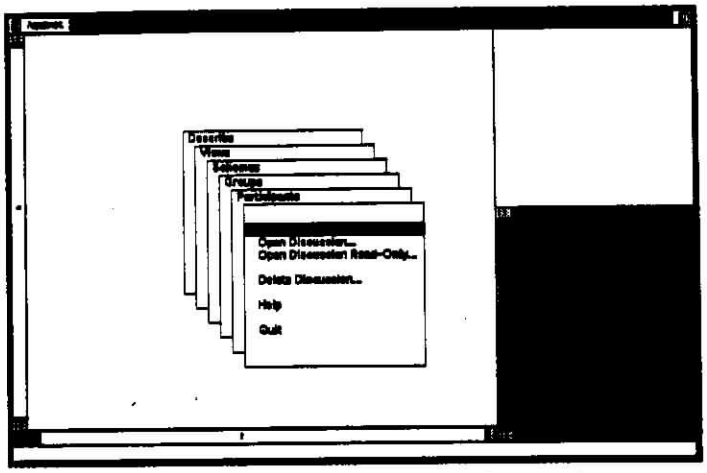


Figure 4.4-1. Starting a new discussion

Naming your discussion and defining who can access it.
When you let go of the mouse button, you will be prompted for the name of your discussion in a dialog box like the one shown in Figure 4.4-2; this name is what other users will see when they try to open a discussion. It will also become part of the window title.

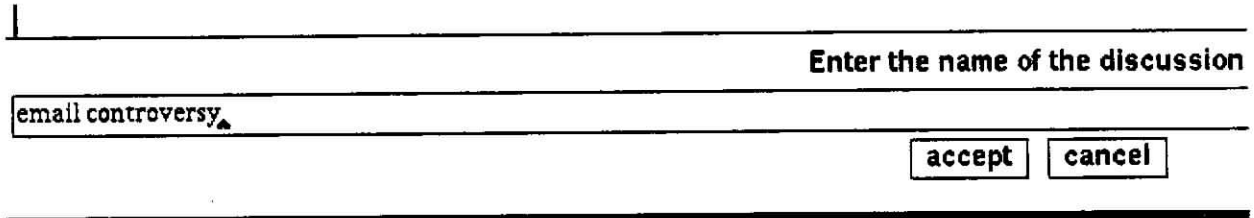


Figure 4.4-2. Naming your discussion

Next you will be asked (via the dialog box shown in Figure 4.4-3) whether you want the discussion to be Unrestricted or Restricted. If you select Unrestricted (the default), then anyone who can run Aquanet can participate in the discussion. If you select Restricted, then you will need to use Participants submenu commands to designate which groups or individuals can participate.

Choosing a schema.
Next you will prompted to choose a schema on which to base the discussion. The list of defined schemas will appear in the Alternate view in the upper right corner of the Aquanet window (as shown in Figure 4.4-4).

Choose a schema by left clicking on one of the schema names in the list; in Figure

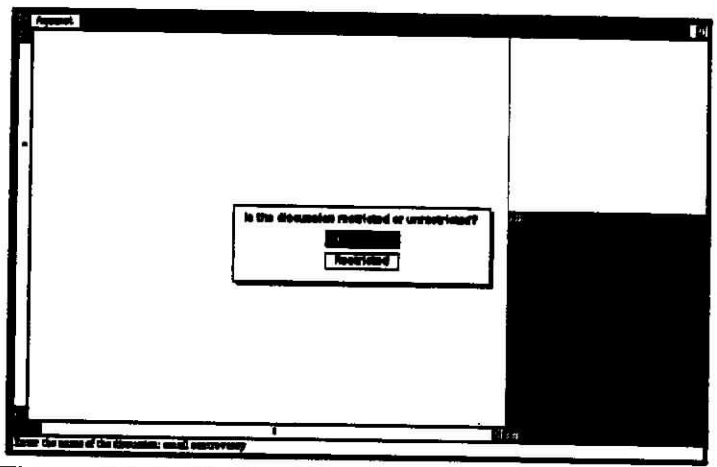


Figure 4.4-3. Controlling access to your discussion

4.4-4, the Simple Arg schema described in Section 1 is being selected for the "email controversy" discussion. If you want to bail out of creating the new discussion, just click on –CANCEL–. If none of the schemas listed are appropriate for your discussion, you may want to define a new one. This process is described in the next section.

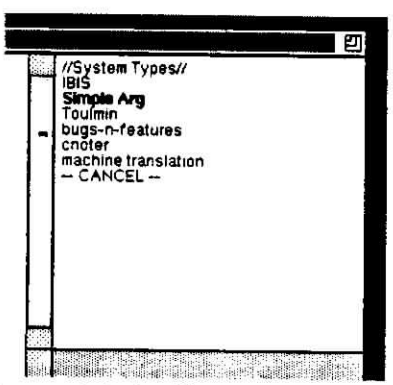


Figure 4.4-4. Choosing a schema

Starting the discussion.
To start a discussion, you can begin creating objects and relations based on the schema you have selected. The Create menu is a submenu of the Discussion menu that you get when you depress the middle mouse button. Figure 4.4-5 shows this submenu as it would appear if the Simple Arg schema were selected. Note that the two top choices are the ***basic objects*** of the Simple Arg schema, Statement (an assertion which can be used in a number of different roles) and Note (an annotation or meta-level comment); the bottom two choices on the list, Argument and CounterArg, are the ***relations*** defined in the Simple Arg schema. In the figure, a Statement is being created.

After the object has been created, it will appear in the main view where you last

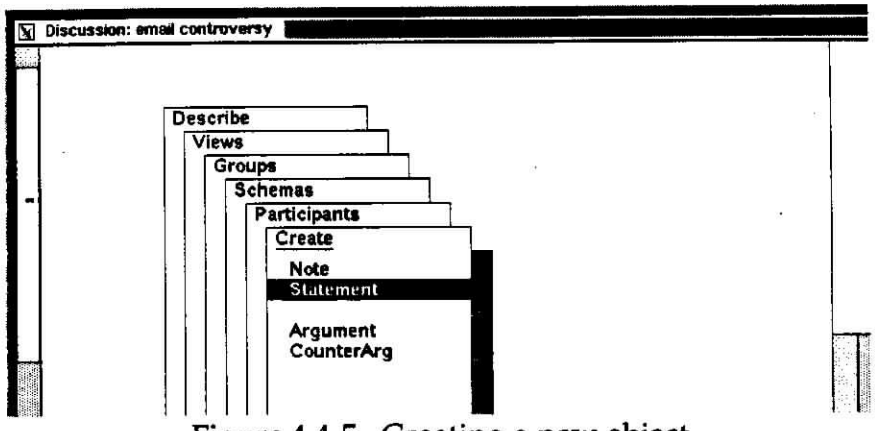


Figure 4.4-5. Creating a new object

left-clicked. If you click on the object with the left mouse button, you have selected the object; it will be highlighted on the display (in red on a color display; in a gray shade on a black and white display) and its contents will be shown in the Node content view. You can move the object by selecting it and dragging it with the left mouse button. In Figure 4.4-6, the Statement we created in Figure 4.4-5 has been selected. Its contents (as yet empty) are shown in the pane on the lower right.

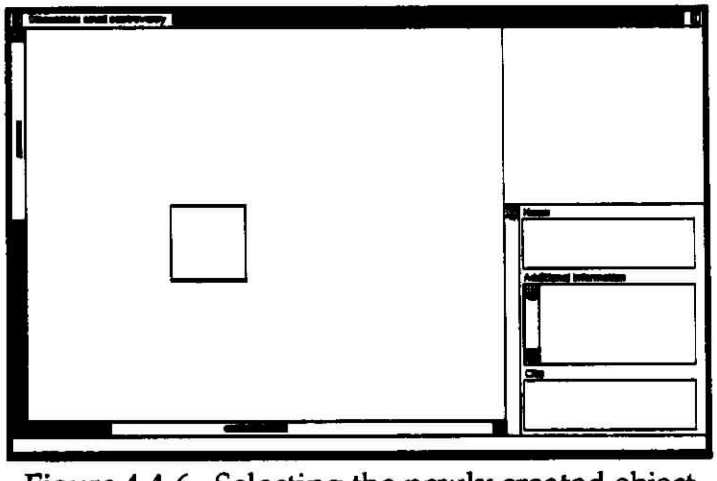


Figure 4.4-6. Selecting the newly created object

Editing an object.
To edit the object, you must first left click in the Node content view to activate it. Then, by pressing the middle mouse button, you can get the menu that will allow you to start editing (see Figure 4.4-7). By selecting the Check Out option on this menu, you will lock the node; this prevents other users from editing a node simultaneously. After you have selected Check Out, left click in the slot you want to edit. A caret will appear, and you can begin typing.

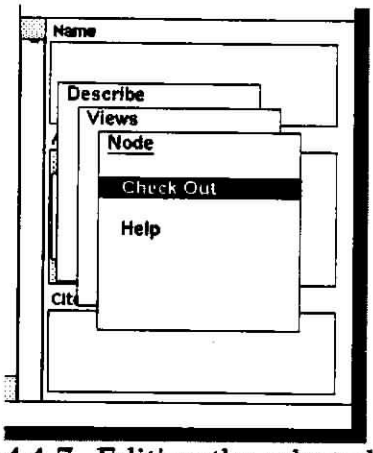


Figure 4.4-7. Editing the selected object

To release the node when you are done editing its contents, left click anywhere in the main view. It will change to reflect the modifications to its contents. In Figure 4.4-8, the Statement's name has changed to reflect the contents of the Name slot in the Node content view. Releasing the node also unlocks it so other users can edit it.

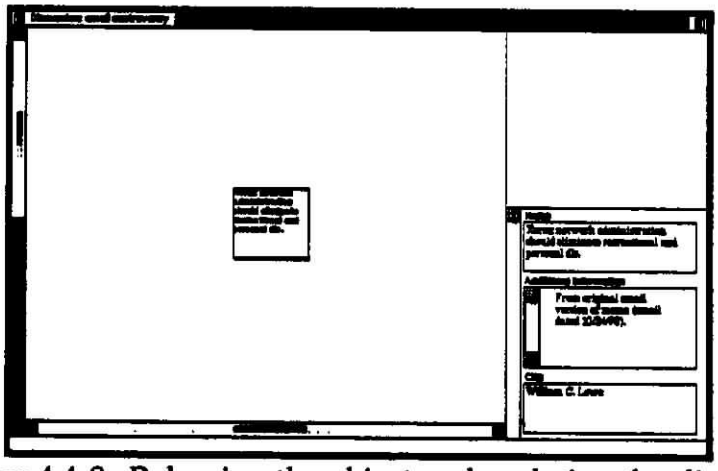


Figure 4.4-8. Releasing the object and updating the display

Creating and filling a new relation.
Creating a new relation is similar to creating a new basic object. Figure 4.4-9 shows a new Argument relation being created using the Simple Arg schema. As before, you get this menu by pressing the middle mouse button and sliding over to the Create submenu.

After you have selected the relation, you can fill its slots. In this example, we'd like to use the statement "Xerox network administration should eliminate recreational and personal dls" as a Conclusion in the Argument relation we have created. Figure 4.4-10 shows the options that will appear

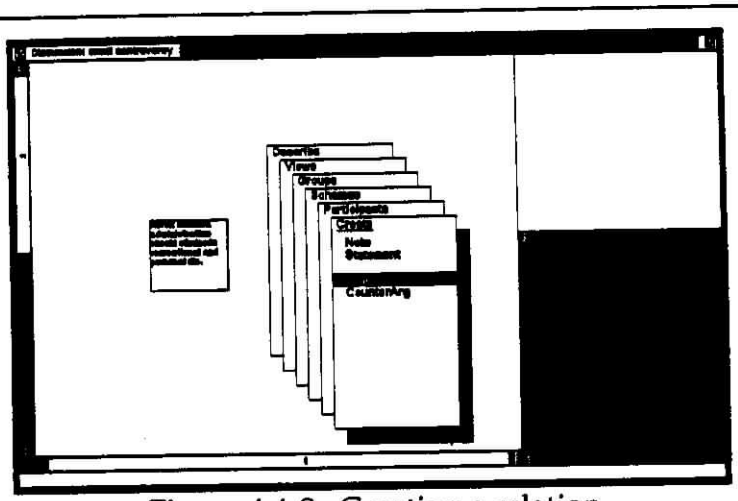


Figure 4.4-9. Creating a relation

when you press the middle mouse button and pull over to get the Slots submenu. Since we already have created the statement that will be used in the slot, we select Fill.

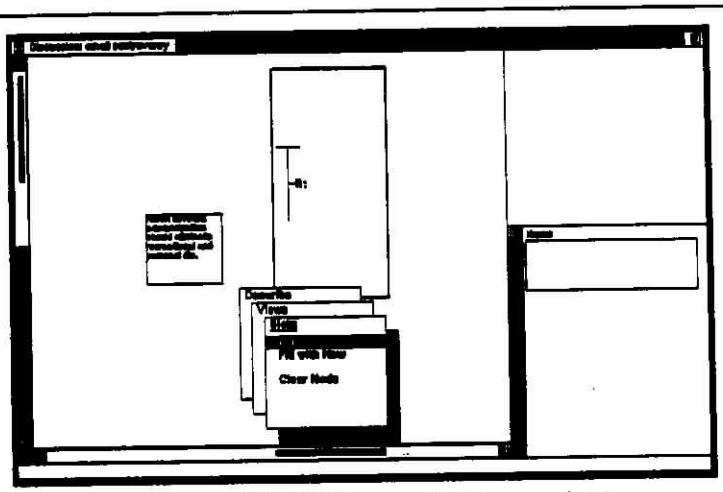


Figure 4.4-10. Filling a slot in a relation

Fill will use a dialog box to ask you which slot you want to fill (unless the relation only has one kind of slot). Figure 4.4-11 shows the dialog box for filling the Argument relation.

Then Fill will prompt you for a node to fill it with. In this case, we have selected the Statement we created earlier. Figure 4.4-12 shows the results of this operation. Note that there is now a virtual copy of the statement filling the slot. If either the basic object or its virtual copy are selected, both will be highlighted.

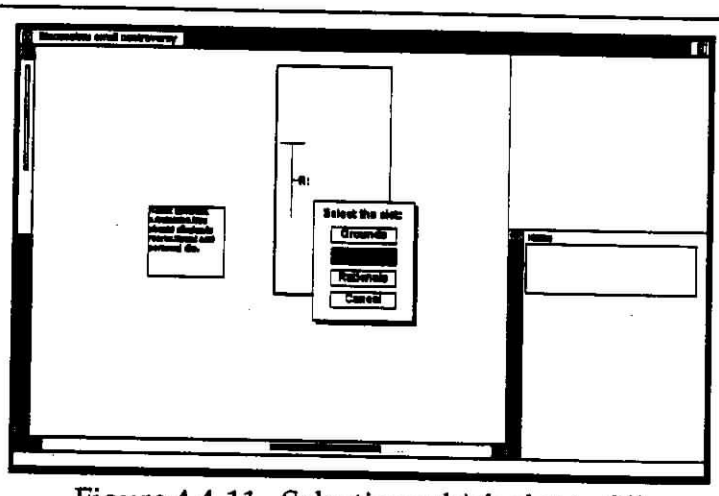


Figure 4.4-11. Selecting which slot to fill

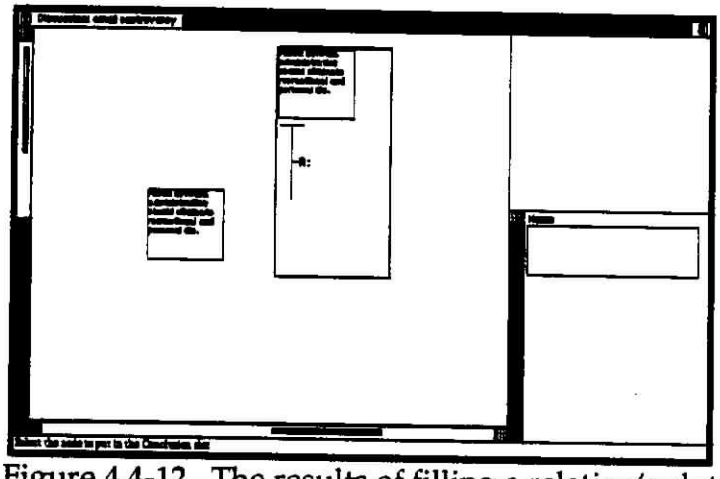


Figure 4.4-12. The results of filling a relation's slot

The other way to fill slots in a relation is to use the Fill with New option on the Slots submenu. When you select this option, Aquanet will ask you to select which slot you want to fill, then it will create an object of the appropriate type (if there is only one appropriate type) and put it in the slot. If there are multiple types of basic objects that can fill the slot, Aquanet will put up a dialog box so that you can indicate which type of object you'd like to put in the slot.

In this way, a relation can be filled out with objects. Figure 4.4-13 shows a (mostly) filled relation.

Building up networks.

Once you have created one relation, you may want to connect it with others. In Aquanet, relations are connected by sharing elements. In the example we have been using, suppose we want to support the Argument's Grounds

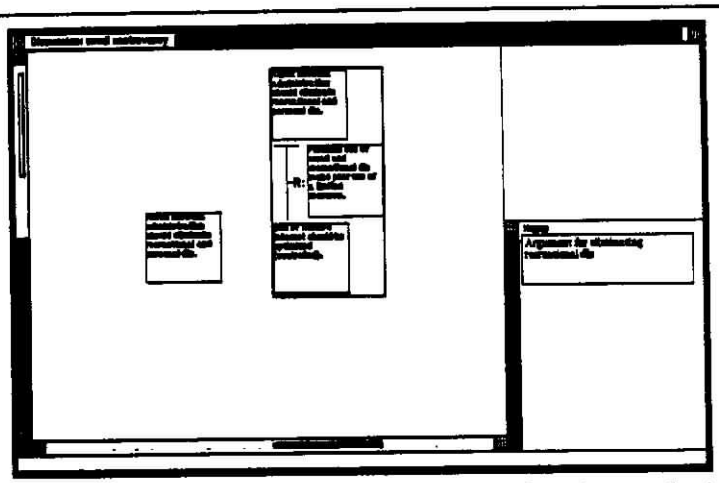


Figure 4.4-13. The results of filling in slots in a relation

"Use of Xerox's Internet should be optimized (controlled)" by making it the Conclusion of another Argument relation.

To do this, we would select the Statement that is acting as the Grounds in the existing Argument. Pressing the middle mouse button down, and pulling over to the Expand As submenu gives us the options shown in Figure 4.4-14. Note that we can use the Datum in a variety of roles in the new relation that will be created as a result of this operation.

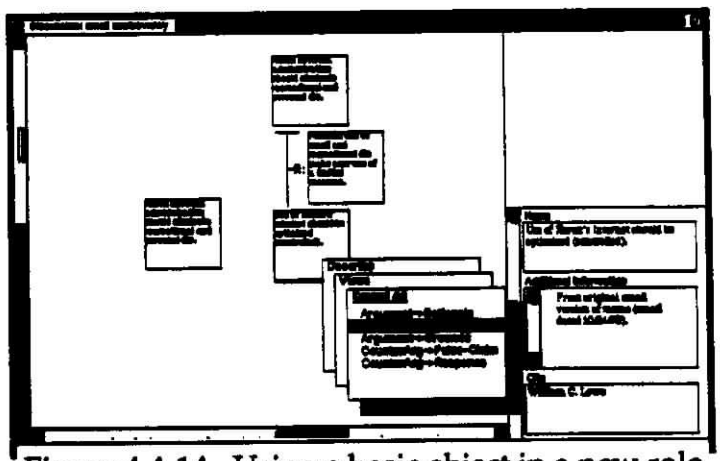


Figure 4.4-14. Using a basic object in a new role

Figure 4.4-15 shows the result this operation. A new Argument relation has been created with the Grounds of the first relation being used as its Conclusion.

To complete the example in this section, the two interconnected Arguments have been filled with statements that duplicate those in Figure 1.3-4. Figure 4.4-16 shows this structure as a Aquanet network.

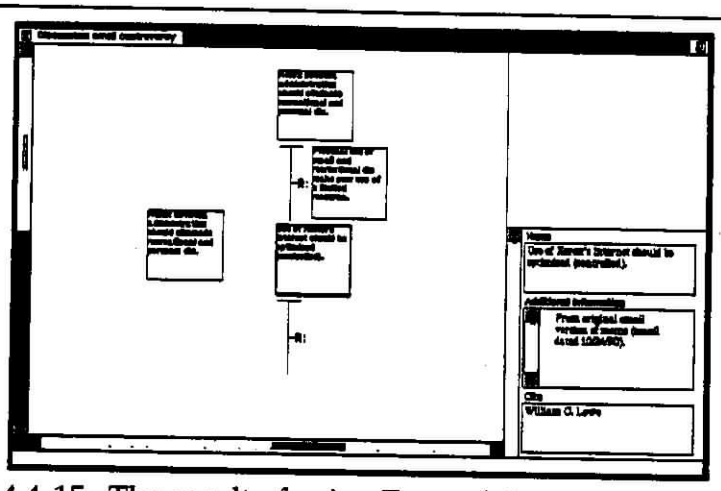


Figure 4.4-15. The result of using Expand As to create a network

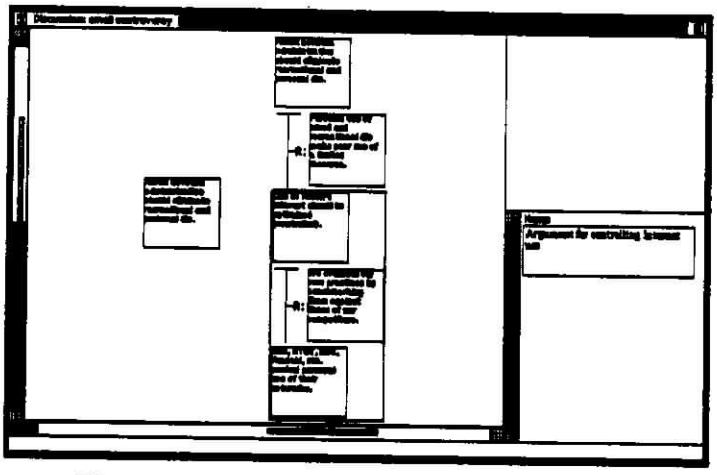


Figure 4.4-16. A growing Aquanet network

4.5 Creating a schema and editing its appearance.

In Aquanet, discussions are always structured by schemas; the schemas can be complex descriptions of a knowledge domain or a process, or they can be very simple and used as a means to organize unexplored territory. The schema editor allows you to define schemas and modify them as your understanding evolves.

Schemas consist of a group of types; the types may be basic objects or relations that specify how basic objects can be connected into a network. This section discusses how to use the main schema editing window and its types editor to create a schema from scratch or modify an existing schema. This section assumes that you have brought up the schema editor on your display by choosing either Create or Edit from the Schema submenu in the main view.

The Schema Editor.
The schema editor is simply a list of the types that have been defined for the

schema and some buttons for controlling the schema editor; Figure 4.5-1 shows the schema editor.

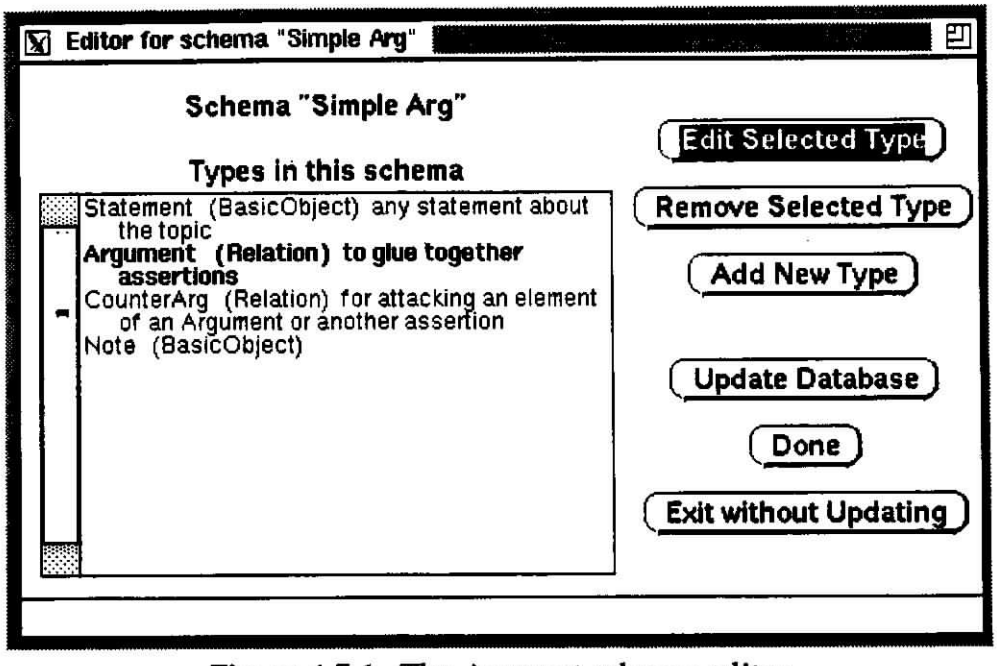


Figure 4.5-1. The Aquanet schema editor

The list on the left side of the window tells you the names of the schema's types, whether each is a basic object or a relation, and any comments describing its use. Any of these types can be selected by left clicking on the type (selection is indicated to you by bold).

The top set of buttons on the right side of the window controls access to the types editor (described in the next subsection). Edit Selected Type causes the type editor to be brought up on the selected type; the type editor will appear in its own window. Remove Selected Type removes the selected type from this schema. Add New Type brings up a fresh type editor window.

The bottom set of buttons on the right side of the window allows you to exit the schema editor, and possibly update the database. Update Database writes the schema changes that you have made thusfar to the database; it does not exit from the schema editor; Done saves your changes to the database and exits from the schema editor; and Exit without Updating flushes the changes you've made since the last Update Database and exits from the schema editor.

The Type Editor.

The type editor (shown in Figure 4.5-2) can be divided by its four basic functions, indicated in the figure by dashed lines. First it allows you to define the name and basic properties of the type. Second, it allows you to specify Supertypes that the type is to be modelled after. Third, it allows you to define the type's slots. Finally,

it enables you to specify the type's graphic appearance.

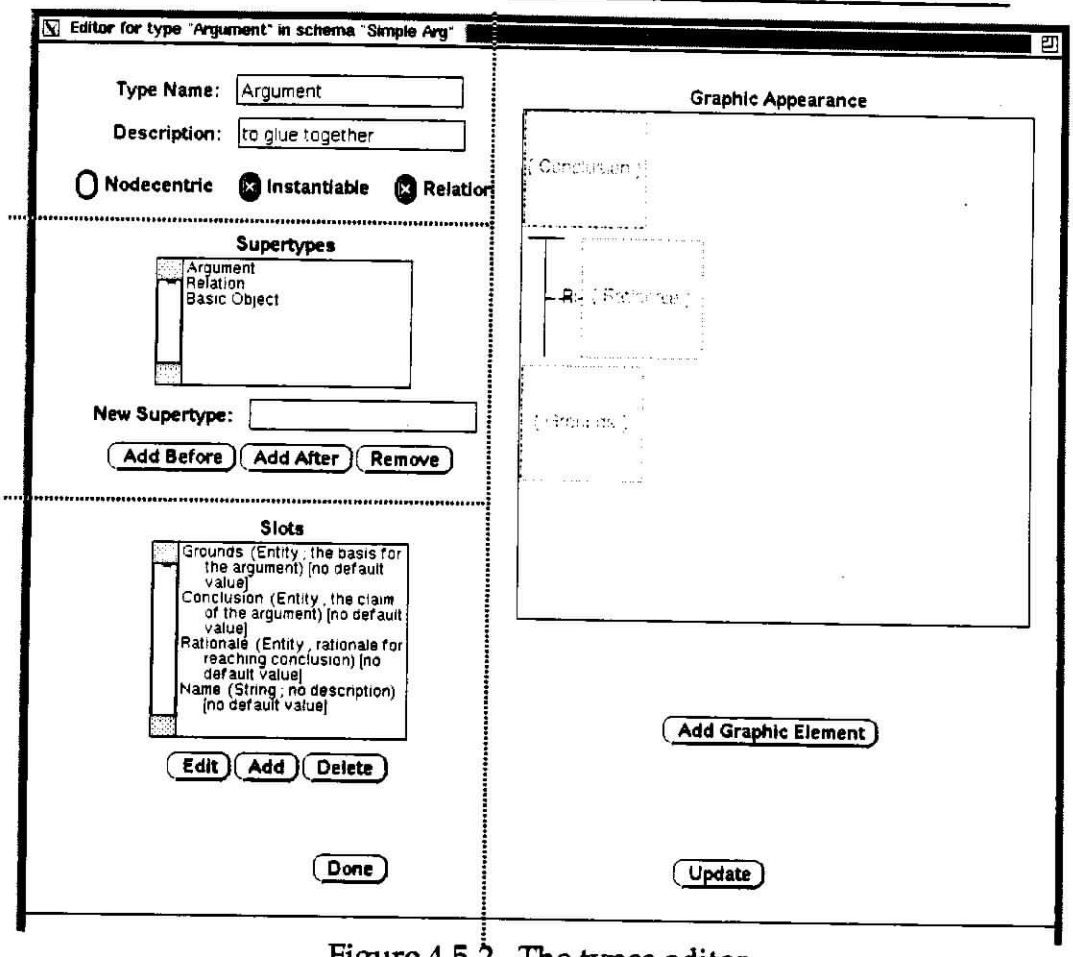


Figure 4.5-2. The types editor.

The upper left-hand area of the type editor provides two text input regions. The top one allows you to name the type (if you are creating a new one), or change the name of an existing type. The lower text input region allows you to write a short comment about the type's purpose. Below these two text input regions are three toggles. Nodecentric, if it is toggled on (indicated by the toggle switch turning black) specifies that this type (which must be a relation) uses the nodes it contains in its slots to determine its layout. Instantiable simply tells Aquanet whether users are allowed to create this type within a discussion - if Instantiable is toggled on, then the type will appear on the Create submenu. The third toggle allows you to specify whether the type is a basic object or a relation.

The middle left region allows you to specify Supertypes for a type - one or more types which the selected type inherits slots from. You must type the name of the new Supertype in the text input region, then specify via the buttons below it whether you want it added before or after the selected type. You can also remove a Supertype from the list. Supertypes are an important shortcut that allow a new

type to inherit properties such as slots and behavior from existing types. For example, if you wanted to create a second kind of statement, a specialization called an Assertion, you can type in Statement to use as a supertype.

All basic objects inherit behavior from the system type Basic Object. Similarly, all relations inherit behavior from the system type Relation. Naturally, Aquanet will become confused if you give a new type some conflicting Supertypes.

The Slot Editor.
The bottom left region of the type editor allows you to edit, add, or delete a slot from the type (refer back to Figure 4.5-2). The scrollable list contains all the slots associated with the selected type. The Name p-slot is supplied automatically by the system when you create a new type. The others are all added through the schema editor. Each slot is displayed with a short description (which the slot's creator supplies) and a default value (if there is one).

When you have selected a slot name from the scrollable list and clicked on the Edit button, the display changes to look like Figure 4.5-3. It will also change to this form if you have no slot name, selected and click on the Add button. Using this slot editor, you can name the slot (the top text input field), flip through the available types for a slot (only if you're adding a new slot - Aquanet does not currently support changing the type of a slot), specify the types of nodes that are allowed to fill Entity-valued slots (e-slots), and comment on the purpose of the slot.

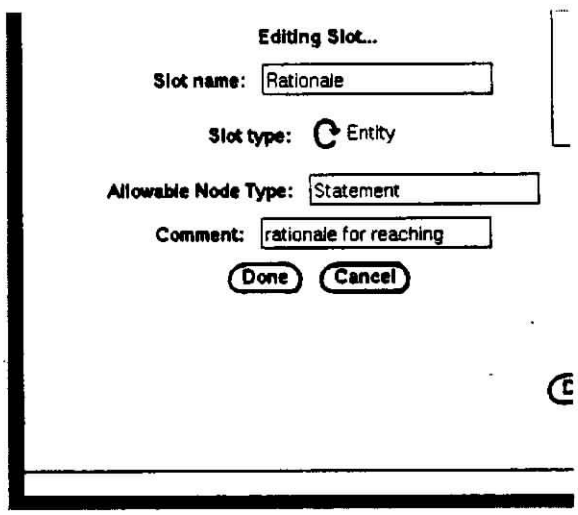


Figure 4.5-3 Editing a slot

The slot types you will see when you click on the circular arrow are Integer, Real, String, Text, Date, Money, Collection, Entity, and Boolean. Integer and Real are numerically-valued types. Strings allow you to store limited-length pieces of text (<255 characters). The Text type allows you to store text of unlimited length in the slot. Dates, Money, and Collection are special Aquanet datatypes. A slot of type

Boolean can only contain values 1 and 0, interpreted as TRUE and FALSE, respectively. Entity-valued slots are only permitted in relations; they will display whatever node has been used to fill them.

If you have specified that the slot is an e-slot (by clicking the Chooser until Entity shows up as the Slot type), a text input area will appear so you can specify what node types can fill the slot. This is where you constrain what types in your schema can be used to fill a slot in a relation. Note that in this case, we've said that the Rationale slot can only be filled with a Statement.

When you have completed editing the slot, clicking on either Done (to save your changes) or Cancel (to cancel your changes) will return you to the original Types Editor display. If you've clicked Done, your changes are not yet written to the database; they are only written to the database after you click on either Update Database or Done in the main schema editor (see Figure 4.5-1).

The Graphic Appearance Editor.

The right half of the type editor window allows you to describe a type's graphic appearance. This is done through direct manipulation of graphic elements in the graphic appearance editor (consisting of the large rectangular area in Figure 4.5-4 labelled "Grapic Appearance" and the control panel beneath it). To get the full graphic editing control panel, select one of the graphic elements in the appearance display by clicking on it with the left button or sweeping out a region around it by holding down the left button.

In Figure 4.5-4, the graphic appearance editor contains the graphic elements of the Argument relation used in the examples in earlier sections. Note that it consists of the regions where the entity-valued slots appear, the three lines that connect them, and a piece of fixed text (an "R:"). In Figure 4.5-4, we have selected the vertical line for editing. Notice that the area below the Graphic Appearance display changes to reflect the kinds of things you can change about Fixed Lines..

To add a new graphic element, select Add Graphic Element to get into the appearance editor (rather than selecting an existing element). The cursor will turn into crosshairs so you can sweep out the bounding box for the new element. Using the top button that appears, you can specify what kind of graphic element you want to add to the type's appearance - Rectangle, Oval, Circle, Fixed Line, Text, or Value of a slot. The other options will change accordingly.

If the option that you've picked for graphic element type is Value of a slot, click on the Change Slot button to specify which slot should be displayed in the area you've swept out. You will be prompted to click on the desired slot from the list displayed in the lower left corner of the type editor. In our example, a Statement displays its Name slot (and a second element, a black rectangle). Figure 4.5-5 shows the graphical appearance editor brought up on this element.

When you're adding or editing a graphic element, after you have selected a set of

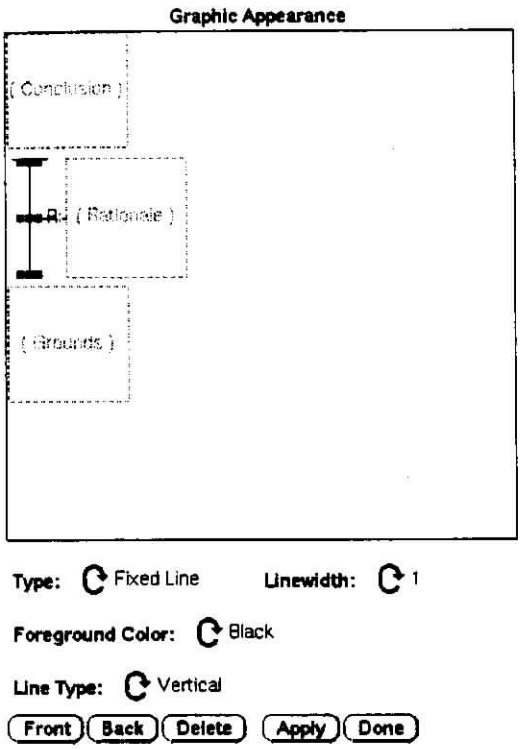


Figure 4.5-4 Editing a graphic element

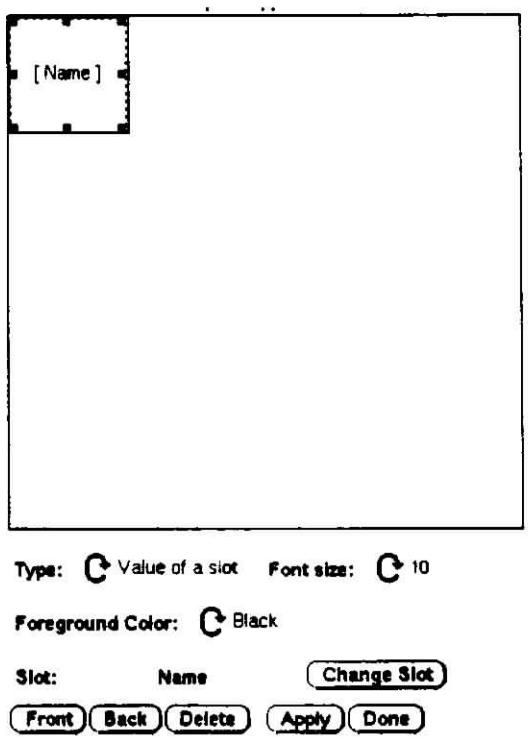


Figure 4.5-5. The graphic appearance of a Slot Value

acceptable options, you can cause the type's graphical appearance to reflect those options by clicking the Apply button. When you are done editing the element, click Done.

If you want to change the shape of an element, select it in the Graphic Appearance display, and use its "knots" (see Figure 4.5-6) to reshape it. The knots at any of the four corners allow you to stretch the graphic in both directions at once (thus reshaping it); by contrast, use the knots in the middle of any of the sides to resize only one dimension at a time. The knot in the middle of the graphic is used for dragging the graphic to another position without changing its shape.

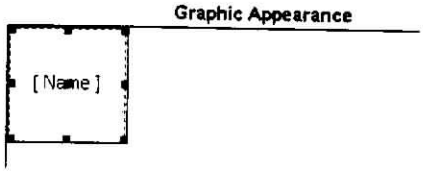


Figure 4.5-6. A graphic element and its knots

After you have completed editing a particular type, click the Done button at the bottom of the types editor; this will return you to the main schema editor (shown in Figure 4.5-1). At this point, you can either edit another type, Update the database with the changes to the type you've just edited (advisable, even if you are about to edit another type, since you may not want to keep the changes you've made to that one), exit the Schema Editor saving your changes (by clicking Done), or exit the schema editor without saving any changes that you've made since the last update.

5. Aquanet Command Summary

This section contains a context-by-context overview of commands. It also lists key bindings that work as command shortcuts.

5.1 Aquanet commands.

This section describes all the middle button commands available in Aquanet.

Aquanet startup menu.
This is the menu that you will see when you start Aquanet and press the middle mouse button anywhere in the main view. The menu has five Aquanet submenus, Discussion commands (unlabeled), Participants, Groups, Schemas, and Views. The commands on these menus enable you to set up and participate in discussions, define who else can participate, delimit groups to control discussion access, create and modify the schemas, and start new views in the Alternate view pane. The startup menu also includes the Describe submenu; it allows you to find out some information about the system.

Discussion submenu.
With Discussion commands, you can enter and participate in an on-going discussion, browse a discussion without contributing, or create a new one. You can also deactivate any discussions you've started, access a help file that describes startup menu commands (similar to the one you are now reading), or quit from your Aquanet session.

New Discussion... New Discussion allows you to initiate a new discussion. It will ask you for its name in a dialog box.

Aquanet will then list all existing schemas in the Alternate view, and request that you select one of them to use as a basis for the new discussion. Note that if you want to base the new discussion on a new schema, you must create the schema first, using the Create option on the Schema submenu.

After you have selected a schema, a dialog box will appear to ask you whether the discussion is Unrestricted or Restricted. An Unrestricted discussion means that anyone who can run Aquanet can participate in your discussion, while restricting a discussion allows you to limit participation to a specific group or individuals. If you have restricted the discussion, you will need to use the Participants submenu to specify who has access.

Open Discussion... Open Discussion allows you to participate in any existing discussion that you have access to (either by being a member of the discussion group or by virtue of the discussion being open to the community at large). The command shows you a list of

all such discussions in the Alternate view of the Aquanet window. Use the left mouse button to select the discussion you want to participate in. After a brief wait while the system consults the database, Open Discussion will display the current version of the network in the main view.

Open Discussion Read-Only... Open Discussion Read-Only allows you to browse any existing discussion that you have read access to. The command shows you a list of all such discussions in the Alternate view of the Aquanet window. Use the left mouse button to select the discussion you want to browse. After a brief wait while the system consults the database, Open Discussion Read-Only will display the current version of the network in the main view.

Delete Discussion... Delete Discussion makes a selected discussion inaccessible. Only the discussion's initiator may delete the discussion. When you choose Delete Discussion, you will be offered a list of all existing discussions in the Alternate View. Using the left mouse button, select the one you want to delete. The contents of the discussion are not removed from the database, but it will no longer appear in the list of active discussions.

Help. Help will present this material to you on-line in an Andrew Help window.

Quit. Quit ends the Aquanet session and closes its window. When you select Quit, a dialog box will appear to check whether you really meant to exit Aquanet; Cancel returns you to Aquanet.

All changes to the discussion are saved as they are made (or as nodes are checked in), so quitting just ends the session.

Participants submenu.
The Participants submenu contains commands for controlling access to a particular discussion.

Add Participant. First, Add Participant prompts you to select which discussion you want to add a participant to; the list of discussions is displayed in the Alternate view. Then it lists all groups and users so you can select whom you want to add to the discussion. Finally, it asks you (in a dialog box) to specify the access level (Read Only, or Read/Write) of the new participant. You can only add participants to discussions that you have created.

Change Access. Change Access allows you (as the owner of a discussion) to change a discussion participant's access to a discussion from read access to read/write access and vice versa. You

will be prompted for the discussion, the member, and the new access level.

Remove Participant. Remove Participant allows you to remove a current participant from a discussion (when all other means of social control have failed). Remove Participant will prompt you for the discussion and member or group; both selection lists will appear in the Alternate View.

Change Restriction. Change Restriction allows you to switch a discussion from restricted to unrestricted, and vice-versa. You will be asked which discussion you want to change the restriction on, then you will be given a choice of the two options (unrestricted and restricted) in a dialog box. If you have changed an unrestricted discussion to be restricted, you will need to use the Participants sub menu to designate which people and groups now have access. Only the person who created the discussion can change its restriction.

Groups submenu.
Group commands define and modify the membership of groups that serve as access control for discussions. Using Group commands, you can modify who can access particular discussions, or you can create a new group for the purpose of restricting access to a discussion you are initiating.

Add Member. Add Member displays a list of all the active groups in the Alternate view. Use the left mouse button to select the one you wish to add a new member to. The system then displays a list of current Aquanet users so you can select which one you want to add.

You can only add members to groups if you created the group. Since Aquanet maintains its own tables of users and groups, you can define groups and their members freely without changing their status in any outside (e.g. Unix or NS) groups. User names correspond to the user's login name.

Remove Member. Remove Member displays a list of all the active groups in the Alternate view. Use the left mouse button to select the one you wish to remove a member from; the system will display the names of all the members in the Alternate view. Again, use the left mouse button to select the appropriate entry. Like Add Member, Delete Member requires that you have the authority to perform the requested operation. Thus you cannot use Remove member as a secondary argumentation strategy.

Create Group. Create Group creates a new active group. This command is useful if you want to initiate a restricted discussion, and want to define who can participate in the discussion. Create Group

just allows you to name the new group; you will need to define its membership by using Add Member.

Delete Group. Delete Group deactivates a group; it lists all active groups in the Alternate view. Use the left mouse button to select any group you have created from the list. This command is useful when a discussion is over or has been deleted, and you no longer need the group you have defined.

Schema submenu.
Schema commands bring up the Schema Editor (in its own window, which you will be prompted to place), so you can edit an existing schema or define a new one.

Create Schema... Create uses a dialog box to prompt you for the name of a new schema, then brings up the Schema Editor on it. See Schema Editor commands for further details on editing schemas.

Edit Schema... Edit lists existing schemas in the Alternate view. After you use the left mouse button to select one, the system will brings up the Schema Editor on the chosen schema. See Schema Editor commands for further details on editing schemas.

Views submenu.
Views commands provide you with alternate views of the discussion.

Node List. Node List turns on a feature which will give you a list of the names of all nodes in the current discussion. The list will appear in the Alternate View; list elements are selectable and the node contents will appear in the Node content view.

User List. User List displays a list of who is currently using Aquanet and what discussion they have open.

Describe submenu.
The Describe submenu appears at the back of every group of Aquanet menus. Describe commands provide information about the Aquanet system. Only the two most useful of the options are covered here.

Show Key Binding. Show Key Binding will tell you what menu command a particular keystroke corresponds to. You will be prompted to type the key(s) you are interested in. Show Key Binding is context dependent; thus it will only show you the key bindings relevant to the particular situation you are in.

Show Bound Keys. Show Bound Keys will bring up a new window that shows you a list of which keys correspond to which menu

options. Once again, this option will only give you the bindings that pertain to the set of menus you see in this context. A complete list of key bindings for Aquanet commands appears in Section 5.4.

Main view menu.
This is the menu you will see after you enter a discussion and press the middle mouse button anywhere in the Aquanet window. The menu has six submenus, one (unlabeled) containing Discussion and overview display options, one that allows you to create new objects and relations, and the Participants, Groups, Schemas, and Views submenus. As always, the Description submenu is also available so you can display information about the Aquanet system.

Discussion submenu.
By using the Discussion commands, you can change the status of the current discussion, ensure that you have the discussion's current state, or change your perspective on it. You can also quit from the discussion or display the appropriate help file.

Close Discussion. Close Discussion ends your current session in the discussion, allowing you to enter another one. If you are closing the discussion just to exit Aquanet, use the Quit command instead.

Delete Discussion. Delete Discussion makes the current discussion inaccessible. Only the discussion's initiator may delete a discussion. Even after it has been deleted, the discussion and its contents remain in the database.

Update. Update forces your view of the discussion to reflect its current state. Update is useful if you are working on the network at the same time someone else is, and you don't want to wait for the view to update automatically. For example, if you believe that someone has edited the content of the object you're currently reading in the in the Content View, and you'd like to know what the changes are before you respond, you can select Update to get the latest version of the object.

Relayout. Not currently implemented. When it is implemented, it will automatically relayout the network according to the algorithm associated with the schema appearance. Because many schemas have layout properties that make them most readable when done by hand, this option has been left for future implementation.

Zoom. Zoom makes the elements of the network proportionately larger in step sizes of 10 percent. This command allows you to focus on small areas of the network. If you have previously used the Pan Out command to look at a large area of the network, you can use the Zoom command to restore your display.

Pan Out. Pan Out does an Unzoom on the network. That is, it will make the elements of the network proportionately smaller in step sizes of 10 percent. This command enables you to see a larger portion of the whole network. It can be undone by successive uses of the Zoom command.

Help. Help displays this information about main view menu commands in a separate Andrew Help System window. After you have selected Help, you will be prompted to place the new window.

Quit. Quit ends the Aquanet session and closes its window. When you select Quit, a dialog box will appear to check whether you really meant to exit Aquanet; Cancel returns you to Aquanet.

All changes to the discussion are saved as they are made (or as nodes are checked in), so quitting just ends the session.

Create submenu.
Create commands allow you to create new instances of the objects and relations that have been defined by the schema for this discussion.

(varies). This menu will vary according to what objects and relations are defined by the schema your discussion is using. In the argumentation example introduced in Section 1, the Create menu allows you to create one of two Basic Objects - a Note or a Statement - or an empty Argument or Counterargument Relation. You can see an example of this menu in Sections 4.3 and 4.4.

Participants submenu.
The Participants submenu allows you to control access to the current discussion (if you created it).

Add Participant. Add Participant displays a list of all groups and users (in the Alternate View) so you can select whom you want to add to the open discussion. You are also prompted for the new participant's access level.

Change Access. Change Access allows you (as the owner of a discussion) to change a participant's access from read access to read/write access and vice versa. You will be prompted for the member or group and the new access level.

Remove Participant. Remove Participant allows you to remove a participant from the current discussion (when all other means of social control have failed). Remove Participant will prompt you for the name of the individual member or group (in the Alternate View).

Change Restriction. Change Restriction allows you to switch the current discussion between restricted and unrestricted. You will be given a choice of the two options in a dialog box. If you have restricted a discussion, you will need to designate which people and groups now have access. Only the person who created the discussion can change its restriction.

Schemas submenu.
The schema command on this submenu allows you to modify the schema for the current discussion.

Edit Schema. Edit allows you to modify the schema you are using as a basis for this discussion. It will bring up the Aquanet Schema Editor (in its own window) and display the schema for you to modify.

Groups submenu.
This submenu is the same the one that appears on the Aquanet startup menu. Use it to define groups and their membership. Used in the context of an active discussion, it is useful for adding or removing a participant or whole groups of participants.

Views submenu.
Views commands provide you with alternate views of the discussion.

Node List. Node List turns on a feature which will give you a list of the names of all nodes in the current discussion. The list will appear in the Alternate View; list elements are selectable and the node contents will appear in the Node Content View.

User List. User List displays a list of who is currently using Aquanet and what discussion they have open.

Describe submenu.
Describe commands appear in every group of Aquanet menus. They are useful for finding out information about the Aquanet system. The most important of these options (Show Bound Keys) allows you to find out which keystrokes are shortcuts for applying which commands for this set of pop-up menus. These key bindings are also summarized in Section 5.4.

Basic object menu.
This is the menu you will see after you have selected a basic object from either the main or alternate view. It has three submenus, Node (unlabeled), Expand As, and Views, in addition to the usual Describe submenu.

Node submenu (unlabeled).
The Node commands available in this context include two display options,

Top and Bottom, which change the depth of the selected basic object, and a third option, Delete, which removes the basic object from the network. Help displays a help file about basic objects.

Top. Top puts the selected object on top of the other nodes occupying the same x-y area. In other words, if the object is partially obscured by other nodes (objects and relations) displayed in the view, then Topping it puts it in front.

Bottom. If a node is either partially or fully obscuring other objects and relations in the display, then Bottoming it allows you to see the other nodes and relations.

Delete. Delete gives you the opportunity to remove any basic object that you've created from the discussion. Note that it does not work on objects created by others, but only on those you've created.

The Delete command asks you to either confirm or cancel the delete in a dialog box that appears once you have selected the command from the menu.

Delete does not remove all virtual copies of the basic object from the network; it only deletes the specific copy that you have selected.

Help. Help gives you information about basic objects, what they are and what the commands are on the basic object menu, in a separate Andrew Help System window. After you have selected help, click left to place the help window.

Expand As submenu.
Expand As commands allow you to use the selected object in a different role in another (a new) relation.

(varies). The items offered on this submenu will vary according to what objects and relations are defined by the schema your discussion is using. In the Argument example introduced in Section 1, the Expand As menu allows you to use a Statement as either Grounds, Conclusion, or Rationale in an Argument, or as a False-Claim or Response in a CounterArg. When you select one of these options, a new instance of the relation will be created, with the selected basic object (in this case, a Statement) filling the slot you have chosen from the menu.

Describe submenu.
Describe commands appear in every group of Aquanet menus. They are useful for finding out information about the Aquanet system. The most important of these options (Show Bound Keys) allows you to find out

which keystrokes are shortcuts for applying which commands for this set of pop-up menus. These key bindings are also summarized in Section 5.4.

Relation menu.
This is the menu you will see after you have selected a relation from the main or alternate view. It usually has three submenus, Nodes (unlabeled), Slots, and Views, in addition to the standard Describe submenu. It may also have an Expand As submenu if the relation you've selected can be used to fill a slot in another relation.

Node (unlabeled) submenu.
The node commands available for Relations include two display options, Top and Bottom, which change the depth of the selected relation, and a third option, Delete, which removes the relation from the network. Help displays a help file about relations.

Top. Top puts the selected relation on top of the other nodes occupying the same x-y area. In other words, if the relation is partially obscured by other nodes (objects and relations) displayed in the view, then Topping it puts it in front.

Bottom. Similarly, if a relation is partially or completely obscuring another node of interest, the Bottom command causes the selected relation to be in back of the other nodes it is covering. If you suspect there may be a node that you can't see because it is completely obscured by layers of other relations, using the Bottom command several times is a good strategy for finding it.

Delete. Delete allows you to remove any relation that you've created from the discussion. Note that it does not work on relations created by others. Delete asks you to either confirm or cancel the delete in a dialog box.

Help. Help gives you information about relations, what they are and what commands are on the Relation menu, in an Andrew Help System window. After you have selected Help, click left to place the help window

Expand As submenu.
This submenu only appears if you can use the selected relation in a role in another relation. The items offered on this submenu will vary according to the role constraints introduced by the schema used by this discussion.

Slots submenu.
Slots commands allow you to fill the relation's slots with basic objects.

Fill. Fill allows you to fill a slot in the selected relation with an

existing basic object. After you have selected Fill, you will first be requested to choose one of the relation's slots from a list that pops up in a dialog box (if there is more than one type that can be used to fill the slot); you can cancel the operation by clicking CANCEL.

Aquanet then prompts you to click on the node to fill the slot. A copy of the basic object will appear in the slot (in a relation-centric type of relation) or the relation will stretch to encompass the object in the slot (in a node-centric type of relation). If you do not click on an existing node, the slot will remain empty. If you attempt to fill the slot with a node of a different type than the schema specifies, the system will tell you that it could not put the node in the slot.

Fill with New. Fill with New allows you to fill a slot in the selected relation with a new instance of the appropriate type of basic object. The system will display a dialog box so you can select the slot you want to fill; you can cancel the operation at this point.

Aquanet creates an instance of the appropriate type of object, and displays it in the slot (as specified in the relation's description in the schema). If the schema used by your discussion specifies more than one possible type of slot filler, a dialog box will appear so you can choose which one to fill the slot with.

Clear Node. Clear Node allows you to remove a node from a slot without deleting the node. It prompts you for the slot to clear. After you have performed the Clear Node operation, the object that used to be in the slot can be moved freely - it is no longer participating in the relation. If you click outside of the relation's slot areas when you are clearing a node, you will cancel the operation.

Describe submenu.
Describe commands appear in every group of Aquanet menus. They are useful for finding out information about the Aquanet system. The most important of these options (Show Bound Keys) allows you to find out which keystrokes are shortcuts for applying which commands for this set of pop-up menus. These key bindings are also summarized in Section 5.4.

Editing Text in the Node content view.
If you have either a relation or basic object (i.e. any node) selected, you can edit its content in the Node content view (the lower right pane of the Aquanet window).

To edit the content of a selected node, first click in the Node content view to activate it, then select Check Out off the middle button menu that appears in that pane. You can now use a simple text editor to make changes in any one of the p-slots that appears in this pane. Note that the pane as a whole has a scroll bar to allow you to see p-slots that are out of the display. Note also text p-slots have a

scroll bar so you can see longer pieces of text. Changes in the slots displayed in the main view will appear after you have stopped editing and have either clicked somewhere else in the Aquanet window, selected another node, or selected Check In from the Node submenu.

The text editor allows you to move the input selection by clicking with the left mouse button. If you hold down the left button or click with the right one, you can extend the current selection. The middle button gives you some text editing commands that are described below. More extensive description of many of these commands can be found in the Andrew EZ text editor documentation.

Node Check Out submenu.
This submenu will appear after you activate the node content view by clicking in it when a node is selected. It enables you to lock the node (check it out for write access) so you can edit it or get help on the text editor. The Views and Describe submenus are also available.

Check Out. Check Out will ensure that you (and only you) have write access to the node. Thus, while you have the node checked out, no-one else can edit it. After you check it back in (either by using the menu option or clicking on something in the main view), the new content will be stored in the database, and other discussion participants will be able to see your changes.

Help. Help displays a help file about editing nodes in Aquanet. The help file will come up in an Andrew help system window.

Node Check In submenu.
This submenu appears on the top of the submenu stack if you click anywhere in the content view that's not in a p-slot's region; otherwise it will appear toward the back of the submenu stack. The Node Check In submenu enables you to reset the node to its original value, save your changes without checking the node back in, check the node back in, or get help on editing a node's content.

Reset. Reset restores the node's original values (the values its p-slots contained when you checked it out). This is useful if you've edited the text and regret what you've done. Choosing Reset will not relinquish the write lock; the node will remain checked out to you.

Save Changes. Save Changes writes your changes to the database without checking the node back in. Thus if any of the other participants of the discussion do an Update (or one is done automatically), your changes will show on their displays. Also, extensive changes can be saved intermittantly this way to checkpoint your work.

Check In. Check In relinquishes the write lock on the node and writes your changes to the database.

Help. Help displays the contents of a help file on editing node content in an Andrew Help System window.

Text editing submenu.
You will see a number of text editing commands when you click in a p-slot's region after you check out a node. The top submenu gives you the ability to cut, copy or paste text, and allows you to strip off layers of formatting. Cut and Copy appear only when text is selected, and Paste appears only when there is something in the "cut buffer" (i.e. text has been previously cut or copied) and there is a point (not a region) selected.

Cut. Cut removes the selected region of text and puts it in a "cut buffer" so it can be pasted elsewhere in the document. Note that if no text is selected, this option does not appear on the menu.

Copy. Copy places a copy of the selected region of text in the "cut buffer" so it can be pasted elsewhere in the document. Note that if no text is selected, this option does not appear on the menu.

Paste. Paste puts the contents of the "cut buffer" at the selected point in the text. Note that if any text is selected, this option does not appear on the menu. Thus if you want to "paste over" some text, you must first remove the text you want to replace, then get the text you want to put there (by either copying or cutting), and then use the Paste command.

Plainer. Plainer removes a "layer" of formatting from the text. That is, if you've made a selection bold, then italic, so you have bold-italic text, Plainer will make the text bold again. Applying plainer a second time will remove the bolding.

Plainest. Plainest removes all formatting from the text. That is, if you've made a selection bold, then italic, so you have bold-italic text, Plainest will make the text plain again.

Search and Spell submenu.
Search and Spell commands let you search for a string in the field or p-slot you've selected, or check the spelling of that field or p-slot. Note that none of these commands work across p-slots. If you want to search all of a node's p-slots for a string, you must perform multiple searches. The commands only work with the active p-slot.

Forward. Forward searches for a string starting at the selected point or region and continuing to the end of the field or p-slot. You will be

prompted for the string in the Aquanet prompt at the bottom of the window.

Backward. Backward searches for a string starting at the selected point or region and continuing to the beginning of the field or p-slot. You will be prompted for the string in the Aquanet prompt at the bottom of the window.

Search Again. Search Again repeats the search starting at the selection point or region. This command is useful if you have multiple occurrences of the string in the active p-slot

Query Replace. Query Replace is a string search-and-replace with a variety of options. You are prompted for the search string, the replace string, and how you want to replace instances of the search string. See the Andrew EZ documentation for a discussion of this command.

Check Spelling. The Check Spelling command invokes the Andrew spelling checker on the text field.

Page submenu.
The Page submenu contains some standard Andrew document editing commands. These commands are not expected to be relevant to most Aquanet uses.

File submenu.
The File submenu mainly allows you to import unformatted text or text that has been formatted with the Andrew text editor.

Insert File. The Insert File command will prompt you for a Unix file name. It will import the text of this file at the selected point.

Describe submenu.
Describe commands appear in every group of Aquanet menus. They are useful for finding out information about the Aquanet system. The most important of these options, Show Bound Keys, allows you to find out which keystrokes are shortcuts for applying which commands for this set of pop-up menus. These key bindings are also summarized in Section 5.4.

Alternate view menus
You can use the commands on the Views submenu that appears in most Aquanet menu stacks to generate an alternate view of the discussion. Currently there are only two views available, a Node List, and a User List. Each has two submenus (in addition to the standard Views and Describe submenus) that let you control the order of the list, and how it is filtered. Since the commands available for the Node List are slightly different than those available for the User List, each is described

separately.

Node List menu.
The menu that appears when you middle-click at the top of the Alternate view containing a node list has two special submenus, Order By and Show.

Order By submenu.
The Order By submenu lets you control the order in which nodes are listed.

Name. Name alphabetizes the nodes according to the value of their name p-slot. Name is a standard p-slot for all nodes.

Create Date. Create Date lists the nodes from oldest to newest.

Edit Date. Edit Date lists the nodes by the date they were last edited, oldest to newest. This is a good way to see the most recent changes to a discussion.

Type. Type lists the nodes by type, thus grouping together nodes of the same type.

Owner. Owner lists the nodes by who owns them (who created them).

Show submenu.
The Show submenu allows you to filter the list.

All Nodes. All Nodes is the default option; it shows you all the nodes in the current discussion.

New Nodes. New Nodes shows you all the nodes that have been created before or after a specific date. You will be prompted to type in the date (the format is specified in the prompt), and whether you're interested in nodes created before this date or after.

Changed Nodes. Changed Nodes shows you any nodes that have been edited before or after a specific date. You will be prompted to type in the date (the format is specified in the prompt) and whether you're interested in nodes edited before this date or after. This is a good way of finding out what's new in a discussion you've been following.

By Type. By Type allows you to specify what type of node you want to see in the list (for example, Argument).

By Owner. By Owner allows you filter the list by user name.

User List menu.
The menu that appears when you middle-click at the top of the Alternate view containing a user list has two special submenus, Show and Order By.

Show submenu.
The Show submenu allows you to filter the list.

All Users. All Users is the default option; it shows you all active Aquanet users (users with any discussion open).

Users of this Discussion. Users of this Discussion just shows you who currently has this discussion open.

Order By submenu.
The Order By submenu lets you control the order in which nodes are listed.

By User. By User alphabetizes the list according to the users' names.

By Discussion. By Discussion alphabetizes the list according to the discussion names.

5.2 Aquanet key bindings.

Key bindings provide you with a shortcut way of invoking Aquanet commands through keystrokes rather than through menus. Figure 5.4-1 shows current Aquanet key bindings. You can see the list of key bindings for the current context by selecting Show Bound Keys off the Describe submenu. To close the key binding list window, select Quit or Delete Window from the middle button menu.

For all contexts:

^X-^C	Exit Aquanet.
^L	Redraw the window.
^H	Help on appropriate context.

For main view:

^C	Close discussion.
^O	Open discussion.
^R	Open discussion read-only.
^U	Force update from database.
<	Zoom (not implemented).
>	Unzoom (also called Pan out - not implemented).

For nodes (basic objects and relations):

^B	Move object to the bottom of the display stack.
^T	Move object to the top of the display stack.

For content view:

^E	Check out node for editing.

Figure 5.3-1. Aquanet key bindings

6. Known bugs and caveats

This section contains a list of bugs and some caveats about features not yet implemented, network-related "effects," and other problems.

6.1 Features not yet implemented.

Aquanet has no undo. Aquanet has no undo for any of its operations. This is clearly a desirable feature for some operations like delete, but it has not been implemented yet.

Aquanet has no "abort session." Aquanet has no way of removing all your changes from a discussion. So if you enter a discussion, make some contributions, and then want to exit, nullifying whatever you've done during the session, you must delete all nodes you've created. Otherwise your contribution will remain in the Aquanet database.

The z dimension (node depth) does not get saved. Aquanet currently does not save the depth of nodes. Thus, if you've arranged a group of overlapping nodes by an artful series of Tops and Bottoms, your arrangement will not be preserved after you've exited the discussion.

Moving nodes does not initiate a lock. Because moving a node may initiate a ripple effect through a large interconnected network of other nodes, it is possible for several users to get into a tug-of-war. If this happens, let the other user move first, since the last move is the one that will be stored in the database. Performance considerations make node-locking on move a prohibitively expensive solution.

Zoom and Pan Out are not implemented. The Zoom and Pan Out options on the Discussion menu are not fully implemented yet. If either is used, you will find that scaling is erratic. However, it's easy to recover if you accidentally either Zoom in or Pan Out since each of the operations exactly reverses the other.

Relayout is not implemented. Relayout is not implemented.

6.2 Side effects.

Loss of database connection. Occasionally you will lose your connection with Sybase, either because the network is flaky or there are too many people trying to access Sybase at the same time, and Aquanet will temporarily freeze.

6.3 Bugs.

Menus are occasionally incomplete. If the menu that appears when you middle click in a window does not correspond to the documentation, just try it again. Sometimes you may have to click left to change the window focus. This is particularly true for the discussion menu.

Drag updates are slow. Sometimes dragging nodes around in the main view can be annoyingly slow. We expect to fix this problem in the bouffant release of the tool.

Relations sometimes don't display their slots in the proper position. Dragging relations a minor distance sometimes will cause slot contents to be "left behind." This is a display bug - the slot contents are intact and will often display properly if you move them again.

If you try to delete a relation you don't own, you will clear its slots. Although you can't delete a relation you don't own, if you try, the nodes filling its slots will be cleared (ie. freed) from the relation.

The schema editor's toggles, choosers, and buttons look strange. If the schema editor's toggles, choosers, and buttons don't look like they do in the figures in this user's guide, you may have a missing font. To get this font, change your font path by typing the following command into your shell window:

```
xset +fp /import/local/andrew/X11fonts
```

Bugs not on this list can be reported to AquanetSupport.parc@xerox.com.

7. The Aquanet Database

This section describes the Aquanet database; full details of the database schema are documented in Appendix A.

Aquanet uses Sybase (a relational database) as its backend to provide reliable storage and concurrancy mechanisms. All Aquanet discussions are stored in a single Sybase database called "Aquanet." The database has twenty-odd tables. Items in the tables are identified by a UID (Unique Identifier). Some also have names which are user-readable. The following list summarizes the tables in the database:

Table	Description
Aquanet Datatypes	Special user-defined types for the Aquanet database schema; this table defines UIDs, types of primitive slot values, and types of change log actions.
Schemas	Basic information about each schema (identifier, name, whether it has been deleted, and help string).
Types	Basic information about each relation and object type, including its identifier, name, whether it's node-centric, width, height, and graphic appearance.
Schema_Types	List of types and included schemas used in each schema.
Type_Supertypes	List of supertypes for each type (to be used by the inheritance mechanism - includes order for multiple inheritance).
Type_Slots	List of both primitive and entity-valued slots used in each type, including the slot's name, the type of values it accepts, and its default value.
Slot_TypeList	List of legal types (by Type_ID) accepted by each entity-valued slot (ESlots, by Slot_ID).
Discussions	Basic information on each discussion, including its identifier, its name, who created it, whether it's restricted or has been deleted, and what schema it uses.
Sessions	Information about a user's session in a discussion, including an identifier for the session, user, aquanet process, hostname, discussion, and window info.
Nodes	Information about each node, including its identifier, identifier for its contents, what discussion it's in, who created it when, and its size and location.
Objects	Information about each object, including its identifier, who created it, its type, its name, when it was created, and when it was last edited.
Objects_<type_id>	Each type of object defines a separate table, each with information specific to the p-slots of objects of this type. The tables are not partitioned by discussion.

Figure 7-1. Database table summary (part 1)

Table	Description
Object_Children	List of nodes in each entity-valued slot in each relation; the table connects slots in a relation with the nodes that fill them.
Node_Children	Similar to Object_Children, except list is of actual (as opposed to virtual copies) nodes in each entity-valued slot.
Next_UIDs	Next available unique identifier (by type) for a schema, discussion, type, slot, node, object, entity (user or group), or session.
Change_Log	List of changes to discussions including time of the transaction, session, type of action performed, what object(s) and discussions were changed.
Locks	This table lists the objects locked for write, including what type of object is locked, its identifier, the session that created the lock, and when it was created
Entities	List of all Aquanet users and discussion groups; table associates unique identifier with each entity (user or group).
Group_Members	List of all discussion groups, including which groups they include and who their members are.
Discussion_Members	Table associates a discussion with the users and groups that have access to them; the list distinguishes between read and write access.
Schema_Visible-Types	Table tells what types appear in the create menu for discussions associated with each schema..

Figure 7-2. Database table summary (part 2)

Associated with the Aquanet database are a set of stored procedures for performing some useful maintenance functions such as purging the change log, purging unreferenced objects, purging incomplete links, deleting discussions, users, or nodes from the database, and creating locks. Appendix A contains a complete list of these procedures along with their arguments.

Several common SQL queries that a user maintaining the Aquanet database might make are also documented in Appendix A.

Index

Symbols

A

B

H

I

K

M

N

S

T

U

V

W

Z

Table: **Aquanet Datatypes** Special user-defined types

User type	Storage type	Description
uid	int	Used for unique identifiers
integer	int	Integer slot types
real	float	Real number slot values
string	varchar(255)	String slot values
freetext	text	Long blocks of text for slot values
boolean	tinyint	Boolean valued slots
monetary	money	Cash valued slots
date	datetime	Date and time slot type
action_type	int	Types of actions in the change log
lock_type	tinyint	Types of locks in the Locks table

Table: **Schemas** Basic per schema info

Column name	Type	Description	Index
Schema_ID	uid	Unique identifier for this schema	
Schema_Name	varchar(255)	User name for this schema	
Owner_ID	uid	UID of entity who created this schema	
Deleted	boolean	Has this schema been deleted?	
Comment	varchar(255)	Help String about this schema	

Table: **Types** Basic per type info

Column name	Type	Description	Index
Type_ID	uid	Unique identifier for this type	
Type_Name	varchar(255)	User name for this type	
Table_Name	varchar(255)	Name of table for type's instances	
IsRelation	bit	Relation or BasicObject?	
IsVisible	bit	User createable vs. hidden type	
IsNodeCentric	bit	Node centric or Relation centric?	
Default_Width	smallInt	The standard initial width	
Default_Height	smallInt	The standard initial height	
Graphic_Appearance	text	Text description of the appearance (documented in this appendix)	
Comment	varchar(255)	Help string about this type	

Table: **Schema_Types** List of types for each schema

Column name	Type	Description	Index
Schema_ID	uid	Unique identifier for a schema	
Type_ID	uid	Unique ID for this type	
Deleted	boolean	Deleted from table?	

Table: **Schema_VisibleTypes** Types visible in Create menu

Column name	Type	Description	Index
Schema_ID	uid	Unique identifier for a schema	
Type_ID	uid	Unique identifier for a type	

Table: **Type_Supertypes** List of supertypes for each type

Column name	Type	Description	Index
Type_ID	uid	Unique identifier for a type	
Supertype_ID	uid	Unique identifier for the supertype	
Precedence	tinyint	Order for multiple inheritence	

Table: **Type_Slots** List of slots for each type

Column name	Type	Description	Index
Type_ID	uid	Unique identifier for a type	C
Slot_ID	uid	Unique identifier for a slot	U/I
Slot_Name	varchar(255)	Name for a slot in this type	
Slot_Type	tinyint	Type of values for this slot where (0=Integer; 1=Real; 2=String; 3=Text; 4=Date; 5=Monetary Unit; 6=Collection; 7=Entity; 8=Boolean; 9=AndyObj; 255=unknown type)	
Default_Value	varchar(255)	Default value for this slot	
Maximum_Value	smallint	Max Nodes for ESlot	
Minimum_Value	smallint	Min Nodes for ESlot	
Comment	varchar(255)	Comment about the Slot	
Deleted	boolean	Is this slot deleted?	

Table: **Slot_TypeList** List of legal types for ESlots

Column name	Type	Description	Index
Slot_ID	uid	Unique identifier for a slot	C
Type_ID	uid	Unique identifier for a type	

Table: **Discussions** Basic per discussion info

Column name	Type	Description	Index
Discussion_ID	uid	Unique identifier for this discussion	U/I
Discussion_Name	varchar(255)	Name for this discussion	U/I
Owner_ID	uid	UID of the creator	C
Schema_ID	uid	Schema this discussion uses	I
IsOpen	bit	Is this discussion restricted?	
Comment	varchar(255)	Comment about the Discussion	
Deleted	boolean	Has this discussion been deleted?	

Table: **Sessions** Discussion Portals

Column name	Type	Description	Index
Session_ID	uid	Unique ID for this sessions	U/I
User_ID	uid	UID from User table	I
Process_Number	smallint	process number of aquanet	
Host_Name	varchar(255)	Host machine name	
Discussion_ID	uid	ID of discussion being viewed	C
X	smallint	X coord of view portal	
Y	smallint	Y coord of view portal	
W	smallint	W coord of view portal	
H	smallint	H coord of view portal	
Start_Time	datetime	When this session started	

Table: **Nodes** Node specific info

Column name	Type	Description	Index
Node_ID	uid	Unique identifier for this node	
Object_ID	uid	The substance object	
Discussion_ID	uid	Discussion this node is in	
Owner_ID	uid	UID of the creator	
X	smallint	X Coord of the node	
Y	smallint	Y Coord of the node	
W	smallint	Width of the node	
H	smallint	Height of the node	
Create_Date	datetime	Date the node was created	
Edit_Date	datetime	Date this node was last edited	
Deleted	boolean	Has this node been deleted?	

Table: **Objects** Fixed per object info

Column name	Type	Description	Index
Object_ID	uid	Unique identifier for this object	
Owner_ID	uid	UID of the creator	
Type_ID	uid	Type of this object	
Name	varchar(255)	Human-readable name of this node	
Create_Date	datetime	Date the object was created	
Edit_Date	datetime	Date this object was last edited	

Table: **Objects_<TYPE_ID>** Type specific per object info

Column name	Type	Description	Index
Object_ID	uid	Unique identifier for this object	
Slot_<Slot_ID>	Slot Type	Value of slot <Slot_ID>	
Slot_<Slot_ID>	Slot Type	Value of slot <Slot_ID>	
...	...	...	

Table: **Object_Children** List of Nodes in Entity Slots

Column name	Type	Description	Index
Object_ID	uid	Unique identifier for this object	
Slot_ID	uid	Unique identifier for the slot	
Child_ID	uid	Unique identifier for the child node	

Table: **Node_Children** List of changes to discussions

Column name	Type	Description	Index
Node_ID	uid	Parent Relation Node Object ID	
Slot_ID	uid	ID for the slot	
Child_ID	uid	ID of the node in the slot	

Table: **Next_UIDs** Next, unused UID values

Column name	Type	Description	Index
UID_Type	tinyint	The type of UID where (0=schema; 1=discussion; 2=type; 3=slot; 4=node; 5=object; 6=entity; 7=session)	
Next_UID	uid	Next available schema UID	

Table: **Change_Log** List of changes to discussions

Column name	Type	Description	Index
Time	datetime	Time (to msec) change was made	
Action	action_type	Type of action where (0=node created; 1=node deleted; 2=node moved; 3=node locked; 4=node lock broken; 5=nodes linked; 6=nodes unlinked; 7=object deleted; 8=object updated; 9=object locked; 10=object lock broken; 11=discussion renamed; 12=discussion deleted; 13=schema deleted; 14=schema renamed; 15=schema edited; 16-21 not used; 22=person joined discussion; 23=person left discussion; 24=person changed view)	
Target_ID	uid	Object affected by action	
Data_ID	uid	Secondary object of change	
Discussion_ID	uid	Discussion affected, zero if global	
Session_ID	uid	ID of the editing session	

Table: **Locks** Object lock table

Column name	Type	Description	Index
Lock_Type	lock_type	Type of locked object where (0=object; 1=node; 2=discussion; 3=schema)	
Target_ID	uid	UID of the locked object	
Session_ID	uid	ID of the session which created the lock	
Lock_Time	datetime	Time the lock was created	

Table: **Entities** Users and Groups

Column name	Type	Description	Index
Name	varchar(255)	Name of the entity	
Entity_ID	uid	Unique identifier of this entity	
Owner_ID	uid	ID of the object owner (self if user)	
IsGroup	bit	Is this entity a group?	
Deleted	boolean	If 1, entity has been deleted	
Last_Login	datetime	When this entity's last session began	

Table: **Group_Members** Members and Sub groups of Groups

Column name	Type	Description	Index
Group_ID	uid	Parent group	
Member_ID	uid	ID of the member (group or user)	

Table: **Discussion_Members** Access List for Discussions

Column name	Type	Description	Index
Discussion_ID	uid	UID of the discussion	
Member_ID	uid	UID of the group	
WriteAccess	tinyint	If TRUE, user has write access to discussion	

Stored DB Procedures:

AddGroup(name:varchar(255), owner:varchar(255), groupID:uid)

Adds a new group 'name' with owner 'owner' to the Entities table if there isn't already one by that name; groupID is set to the uid of the new group (or the uid of the existing group if there already is one by this name).

Return status of 0 - a new group has been created

Return status of 1 - the group already exists

AddNode(sessionID:uid, objID:uid, did:uid, ownerID:uid, x:smallint, y:smallint, w:smallint, h:smallint, nodeID:uid)

Adds a node to the Nodes table given the session ID, object ID, discussion ID, owner ID, the node's x, y coordinates, and its width, and height. Returns the uid for the new node. Also records an entry in the Change Log.

AddObject(sessionID:uid, ownerID:uid, typeID:uid, objID:uid)

Adds an object to the Objects table given the session ID, owner ID, and Aquanet type ID. Returns the uid for the new object.

AddType(sessionID:uid, name:varchar(255), schemaID:uid, isRelation:bit, isVisible:bit, isNodeCentric:bit, width:smallint, height:smallint, comment:varchar(255), typeID:uid)

Creates a new entry in the Types table using the parameters passed in and returns the uid of the new type in TypeID. Sets the new type's graphic appearance to null, updates the the Schema_Types table to associate the new type with the schema it belongs to, and creates the Objects_<object_ID> table.

AddUser(name:varchar(255), userID:uid)

Adds an Aquanet user to the Entities table if a user by the input name doesn't already exist. Returns the uid of the new user (or the uid from the Entities table if the user already exists).

Return status of 0 - new user successfully added

Return status of 1 - user already exists

BreakLock(locktype:lock_type, targetID:uid, sessionID:uid, lockOwner:varchar(255))

If the lock exists, it is broken by deleting its entry from the Locks table; otherwise the name of the lock's owner is returned.

Return status of 0 - lock successfully broken

Return status of 1 - lock not broken

CanFillSlot(parentID:uid, childID:uid, slotID:uid)

Checks whether a node can be put in the specified slot by seeing how many nodes already fill the slot in the Node_Children table, and comparing this number to the maximum number of children allowed for the slot in the Type_Slots table.

Return status of 0 - slot can be filled

Return status of 1 - slot can't be filled

ChangeParticipantAccess(dName:varchar(255), user:varchar(255), access:tinyint, owner:varchar(255))

If the discussion exists and its owner is the access change requestor, the new access for user is written to the Discussion_Members table; if the access is passed as -1, the participant is deleted from the discussion. Users who are not in the table are added.

Return status of 0 - successful completion

Return status of 1 - the discussion doesn't exist

Return status of 2 - requestor is not discussion owner

(owner's name is returned)

CollectGarbage(void)

Calls PurgeChangeLog, PurgeUnreferencedObjects, and PurgeBadLinks.

CompactSystemTables(void)

Removes entries with the Deleted bit set. Also removes unreferenced Types, Supers, Slots, Nodes, and Objects and Discussions without schemas.

CreateDiscussion(dName:varchar(255), sName:varchar(255), ownerID:uid, openOrClosed:bit, did:uid)

Creates a new discussion in Discussions and returns its uid.

Return status of 0 - successful discussion creation

Return status of 1 - discussion by the same name already exists

Return status of 2 - schema chosen for the discussion doesn't exist

CreateSchema(name:varchar(255), comment:varchar(255), schemaID:uid)

Creates a new schema in the Schemas table and returns its uid.

Return status of 0 - successful schema creation

Return status of 1 - schema by this name already exists

(its uid is returned)

DeleteDiscussion(sessionID:uid, dName:varchar(255), owner:varchar(255))

Deletes a discussion from Discussions, and all its nodes from the Nodes table by setting the Deleted bit to 1; members are also deleted from the Discussion_Members table and the Change Log is updated. For the delete to be successful, the discussion must exist, the requestor must own the discussion, and no other users can be accessing the discussion.

Return status of 0 - successful delete

Return status of 1 - requestor is not the discussion's owner
(owner's name returned)

Return status of 2 - discussion doesn't exist

Return status of 3 - others are accessing the discussion; can't delete

DeleteGroup(group:varchar(255), owner:varchar(255))

Deletes a group by removing it from the Entities table and removing its members from the Group_Members table. It also removes the group from any discussions it has access to.

Return status of 0 - successful group deletion

Return status of 1 - requestor is not group owner
(owner's name is returned).

DeleteNode(nodeID:uid, sessionID:uid, breakLock:bit, nodeOwner:varchar(255))

Deletes a node if the requestor is the owner and the node isn't locked; to delete the node, a transaction is initiated where all previous actions concerning the node are removed from the Change Log and a new entry is inserted and the node is removed from the Nodes table.

Return status of 0 - successful node deletion

Return status of 1 - requestor is not node owner

Return status of 2 - node is locked

DeleteSchema(schemaID:uid, sessionID:uid, owner:varchar(255))

Sets the Deleted bit to 1 for this schema in the Schemas table and records action in the Change Log. Requestor must be owner of the schema; otherwise the schema is not deleted and the owner's name is returned.

Return status of 0 - successful schema deletion

Return status of 1 - requestor is not schema owner
(schema not deleted)

Return status of 2 - schema does not exist

DeleteSession(sessionID:uid, user:varchar(255))

Deletes an entry from the Sessions table and updates the Change Log.

Returns the name of session owner if user is not the owner.

Return status of 0 - successful session deletion

Return status of 1 - requestor is not session owner
(session is not deleted)

Return status of 2 - session does not exist

DeleteSlot(typeID:uid, slotID:uid)

Deletes a slot associated with the specified type by setting its Deleted bit to 1 in the Type_Slots table.

Return status of 0 - successful deletion of slot

Return status of 1 - no such slot associated with type

DeleteType(typeID:uid, schemaID:uid, sessionID:uid)

Deletes a type from the specified schema by setting the Deleted bit in Schema_Types to 1.

Return status of 0 - successful deletion of type from schema

Return status of 1 - type does not exist in this schema

DeleteUser(username:varchar(255))

Deletes a user from the Entities table by setting the entry's Deleted bit to 1. Also removes user from all groups he or she belongs to in the Group_Members table.

GetAccessLevel(user:varchar(255), dName:varchar(255))

Gets the specified user's access level to a discussion. Unrestricted discussions are assumed to mean that all users have read/write access, and discussion owners are assumed to have read/write access. For restricted discussions, individual access levels are recorded in the Discussion_Members table.

Return status of 0 - user has no access to the discussion

Return status of 1 - user has read access to the discussion

Return status of 2 - user has read/write access to the discussion

GetDiscussionName(discussionID:uid, name:varchar(255))

Returns the discussion name given its ID by looking it up in the Discussions table.

Return status of 0 - successful lookup of discussion name

Return status of 1 - ID not found in Discussions table

GetDiscussionSchema(dName:varchar(255), did:uid, sName:varchar(255),

sid:uid)

Returns the schema ID, schema name, and discussion name associated with the specified discussion ID.

Return status of 0 - successful completion

Return status of 1 - lookup was unsuccessful

GetNextUID(uidtype:tinyint, nextUID:uid)

Returns a new uid of the specified type.

Return status of 0 - new ID has been successfully generated

Return status of 1-8 - no new UID (return status = the uidtype)

GetObjIDFromNodeID(nodeID:int, objID:int)

Returns an object ID from the Nodes table given its node ID.

GetSlotInfo(parentID:uid, childID:uid, slotID:uid, slotName:varchar(255))

Given the ID of a parent and child node, returns the name and ID of the slot the child is filling.

Return status of 0 - successful query for slot info

Return status of 1 - no such parent/child pair

GetTypeInfo(typeID:uid, name:varchar(255), isRelation:bit, isVisible:bit, isNodeCentric:bit)

Given the ID of a type, returns its name, whether it's a relation, whether it appears on the create menu, and whether it's node centric.

Return status of 0 - successful query for type info

Return status of 1 - no type exists with this ID

GetUserName(userID:uid, name:varchar(255))

Looks up a user's name in the Entities table given a user ID.

Return status of 0 - successful lookup of user name

Return status of 1 - ID not found

InitSession(userID:uid, processID:smallint, hostname:varchar(255), sessionID:uid)

Initializes a session given the user ID, the process ID, and the host name of the user's machine. Generates a new uid for the session and inserts the session information into the Sessions table.

Return status of 0 - successful completion

JoinGroup(member:varchar(255), group:varchar(255), owner:varchar(255))

Adds a new member to the specified group in the Group_Members table,

provided that the group exists and the requestor is the owner of the group.

Return status of 0 - successful addition of member to group

Return status of 1 - requestor is not group owner

(owner's name is returned)

LeaveGroup(member:varchar(255), group:varchar(255), owner:varchar(255))

Removes a member from a group by deleting the member's entry in the Group_Members table.

Return status of 0 - member successfully removed from group

Return status of 1 - requestor is not owner of group

(owner's name is returned)

MakeLock(locktype:lock_type, targetID:uid, sessionID:uid, lockOwner:varchar(255))

Locks a node passed in as targetID by making an entry in the Locks table using the information passed to it. If there's already a lock on this node, MakeLock returns the owner of the lock.

Return status of 0 - node has been successfully locked

Return status of 1 - node locked by another user

(lock owner's name is returned)

PurgeBadLinks(void)

Removes non-existant links from the Node_Children table by checking whether both parent and child nodes exist in the Nodes table.

PurgeChangeLog(void)

Deletes all entries from the ChangeLog table that correspond to sessions that no longer exist.

PurgeUnreferencedObjects(void)

Removes objects that are no longer referenced in the Nodes table from the Objects table.

SetSessionDiscussion(sessionID:uid, dname:varchar(255))

Updates the Sessions table with a new discussion ID. If there is no current discussion (the discussion name has a null value), then a NULL value is put in the Sessions.

Return status of 0 - successful update of Sessions table

Return status of 1 - session does not refer to this discussion

SetSlotInfo(typeID:uid, slotID:uid, name:varchar(255), type:tinyint, minVal:int, maxVal:int, defaultVal:varchar(255),

comment:varchar(255))

Updates the Type_Slots table with new information about a slot, or if the slot ID is passed in as 0, generates a new uid and creates an entry for the slot.

Return status of 0 - successful update of info in the Type_Slots table

Return status of 1 - new slot has been created with info

Return status of 2 - no slot name exists for this slot ID

SetTypeInfo(typeID:uid, name:varchar(255), isRelation:bit, isVisible:bit, isNodeCentric:bit, dWidth:int, dHeight:int, comment:varchar(255))

Updates the Types table with new information about a type (passed as parameters to this proc).

Return status of 0 - successful update of info in the Types table

Return status of 1 - no slot name exists for this type ID

TableHasColumn(table:varchar(255), colName:varchar(255))

Checks to see if table already has a column by this name by consulting the aquanet database's tables.

Return status of 0 - no current column by this name

Return status of 1 - already a column by this name

WriteNode(sessionID:uid, nodeID:uid, x:smallint, y:smallint, w:smallint, h:smallint)

Updates the size and position information for a particular node given its ID. Also records change in the Change Log.

Return status of 0 - successful update of node's position and size

Return status of 1 - node was not found in the Nodes table

Graphic Appearance Format

The graphic appearance for each type is specified by three columns in the Types table. The first two, Default_Width and Default_Height, specify the bounding box of the type (in pixels). The third, Graphic_Appearance, contains detailed specifications of each element of a type's appearance. These specifications are stored as text; the description of each element is terminated by a carriage return.

The following is the per-element specification:

element type, x, y, w, h, parameter 6, parameter 7, parameter 8, parameter 9

Element type is a value from 0-9 where

element type = 0: line;

element type = 1: rectangle;

element type = 2: oval;

element type = 3: circle;

element type = 4: text;

element type = 5: slot value (name of p-slot value to display);

element type = 6: icon;

element type = 7: node slot (name of e-slot value to display);

element type = 8: group;

element type = 9: stretchy line (a line that connects two nodes).

Parameters x, y specify the corner of the element's bounding rectangle expressed as a percentage of the type's Default_Width, Default_Height (also in Types table).

Similarly, w and h are the width and height of the element, again expressed as a percentage of Default_Width, Default_Height. The coordinate system for graphic elements is taken as starting at 0,0 in the upper left hand corner.

The one exception to this specification is element type = 9 (stretchy line), which uses the following parameters in place of x,y,w,h:

slot-name-1, attachment-type-1, slot-name-2, attachment-type-2

Parameters slot-name-1 and slot-name-2 define the e-slots that act as endpoints, and attachment types dictate how the lines will be drawn between the two slots of the relation.

attachment type = 0: no preferred attachment point (the program decides);

attachment type = 1: vertical attachments preferred;

attachment type = 2: horizontal attachments preferred;

attachment type = 3: side attachments preferred;

attachment type = 4: corner attachments preferred.

Parameter 6 is either foreground color (if the element is a line, rectangle, oval, circle, text, or slot value) or the name of an e-slot (if the element type is 7). Foreground color may be any named xcolor (to see possible values, type xcolors at the shell prompt on a color monitor).

Parameter 7 is either the background color of an element (if the element is a rectangle, oval, or circle), the actual text to be displayed (if the element type is 4), or the name of the slot to be drawn (for element type 5). Background color may be any named xcolor or Clear (transparent).

Parameter 8 is line weight (for graphics) or font size for text.

Other parameters may be added for future use; all parameters on a single line correspond to the single element of the type specified by the first parameter.

For example, if you wanted to specify a two pixel thick black rectangle with a transparent background that always appears as the lower right hand quarter of a type, and if the Default_Width = 200 and the Default_Height = 400, the element specification would look like:

```
1,50,50,50,50,black,Clear,2
```

As a further example, the entire Graphic Appearance specification (as stored in the Types table) for the type "Argument" used in the User's Manual is:

```
7,2,1,66,31,Conclusion,0,0,0,0,0
7,2,68,66,31,Grounds,0,0,0,0,0
7,33,34,66,32,Rationale,0,0,0,0,0
0,4,34,24,34,black,0,1,0
0,14,34,14,66,black,0,1,0
0,14,50,20,50,black,0,1,0
4,20,50,15,0,black,R:,12,1,0,0
```

Job Messages

Xerox PostScript version 47.0 revision 17
Copyright (c) 1987, 1988, 1989, 1990, 1991 Xerox Corporation.
- - handling undefined [/lettertray] at position 18498 in /var/spool/mdqs/lock/home/data//29648AB61ef5

Aquanet: a hypertext tool to hold your knowledge in place

Catherine C. Marshall
Frank G. Halasz
Russell A. Rogers
William C. Janssen Jr.

Xerox Palo Alto Research Center
3333 Coyote Hill Rd.
Palo Alto, CA 94304
415-494-4740
marshall.parc@xerox.com
Fax: 415-494-4777

Keywords: representation, graphical knowledge structures, complex relations, collaboration

Abstract:

Hypertext systems have traditionally focused on information management and presentation. In contrast, the Aquanet hypertext system described in this paper is designed to support knowledge structuring tasks. In this paper, we discuss our motivations for developing Aquanet. We then describe the basic concepts underlying the tool and give a brief guided tour of the tool in operation. We close with some brief comments about our initial experiences with the tool in use and some of the directions we see the Aquanet research moving in the near future.

Submitted to Hypertext '91, San Antonio, Texas, December 15-18, 1991

1.0 Introduction

The power of hypertext derives from its dual nature: hypertext is simultaneously a tool for managing and presenting information and a tool for representing the structure inherent in that information. Although this essential duality has been duly noted in the hypertext literature, by far the largest part of the effort in building, using, and researching hypertext remains on the information management and presentation side of the duality. From Memex [Bush45] and Augment [Engl68] through KMS [Aksc88] and Intermedia [Garr85], the main focus has been on hypertext as vehicle for managing large collections of non-linear information and for presenting this information to readers in a manner that does justice to its non-linear nature. The other side of this duality, information (or, if you prefer, knowledge) representation has until very recently been considered more the province of Artificial Intelligence than of Hypertext. With the exception of a few scattered projects such as gIBIS [Conk88] and IDE [Jord89], relatively little attention has been focused on exploring and further developing the underlying representational capabilities of the hypertext data model.

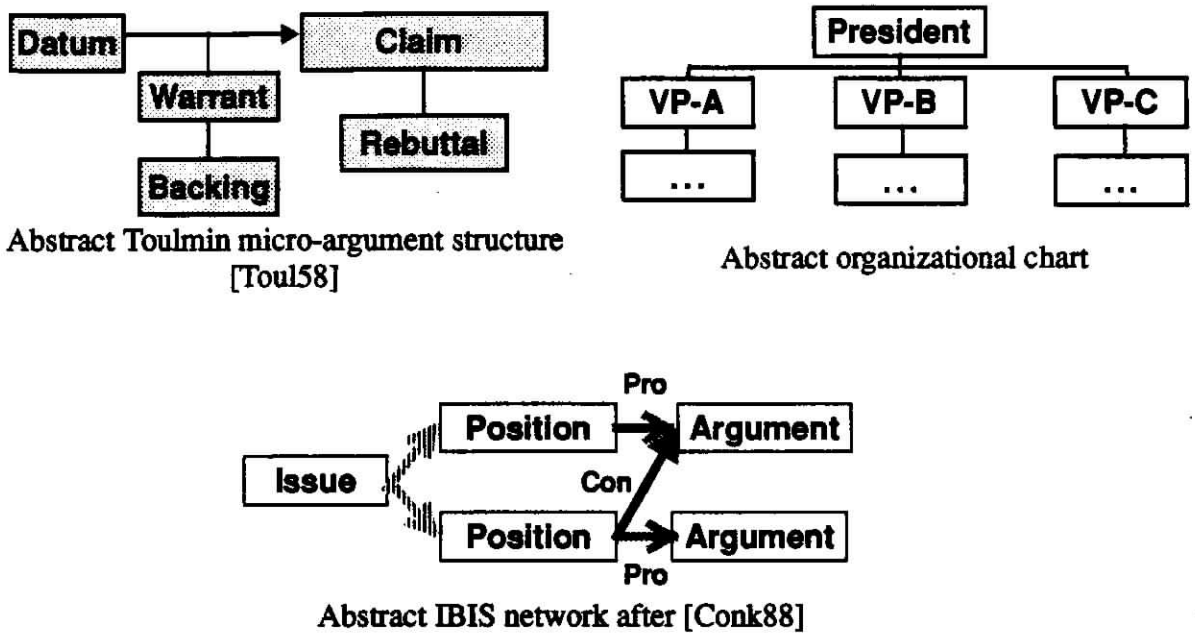
Aquanet is a collaborative hypertext tool for people trying to interpret information and organize their ideas, either individually or in groups; we have been calling such activities *knowledge structuring tasks*. Some knowledge structuring tasks - for example, information analysis - involve movement from unstructured fragments to coherent, organized structures [Brow85][Hala87]. Other knowledge structuring tasks such as design deliberations may use a known structure and the methodology it embodies to facilitate group discussions; Yakemovic's use of IBIS for software design is an excellent example [Yake90].

Our goal in developing Aquanet was to explore the utility of hypertext facilities in the realm of knowledge representation and, as a result, to broaden our understanding of hypertext's representational characteristics. In this paper, we will describe our motivations in developing Aquanet. We will then describe the basic concepts underlying the tool and give a brief guided tour of the tool in operation. We will close with some brief comments about our initial experiences with the tool in use and some of the directions we see the Aquanet research moving in the near future.

Throughout this paper, we will be using the terms "knowledge structure" and "graphical knowledge structure". We use *knowledge structure* to refer to an interconnected network of information-bearing nodes that are used to represent the primitive objects and their interrelationships in some domain of discourse. Technically, any hypertext network can be considered a knowledge structure under this definition. But we intend the term to connote hypertext networks (and other information structures) whose basic purpose is to represent or model the structure of some real-world domain. By *graphical knowledge structure*, we mean a knowledge structure whose primary presentation to the user is as a graphic display on a computer screen.Figure 1 shows three representative examples of the kind of graphical knowledge structures that drove our thinking in developing Aquanet.

2.0 Background: NoteCards meets gIBIS

Aquanet brings together two separate but convergent lines of hypertext research. First, over the last five years we have been observing people using NoteCards [Hala87] for a variety of knowledge structuring tasks. Of particular interest are our investigations into representing the structure of argumentation in hypertext [Mars87][Mars89] and our use of the NoteCards-based IDE system for analyzing knowledge in instructional design tasks [Russ87][Jord89]. Second, we have been



Abstract Toulmin micro-argument structure [Toul58]

Abstract organizational chart

Abstract IBIS network after [Conk88]

Figure 1: Representative knowledge structures

influenced by the ideas introduced in gIBIS [Conk88] and its subsequent generalization, Germ [Brun88]. The gIBIS and Germ tools provide a valuable counterpoint to NoteCards since they were explicitly designed to support the construction and maintenance of constrained knowledge structures.

These two lines of research have highlighted the discrepancies between the needs of users engaged in knowledge structuring tasks and the functionality provided by information management and presentation hypertext systems like Intermedia and KMS. Two discrepancies stand out. First, information management hypertext systems focus on nodes and the local connections between them. But in knowledge structuring tasks it is important to see and manipulate a global view of the network. Second, information management systems provide only a simple node-link data model. Knowledge structuring tasks often require a richer language for expressing the interconnections among nodes.

2.1 Centering interaction on a network overview

One of the most important consequences of emphasizing the information management and presentation aspect of hypertext is that interaction tends to be centered on nodes or documents; authors and readers are expected to focus on the textual or graphic content of nodes, and to move from node to node in navigating through a network. In fact, systems like NLS/Augment [Engl68], KMS, and Hypercard [Appl87] provide no structural overview of the network. By contrast, gIBIS interactions center around a graphic overview of the emerging hypertext network; new nodes and

links are always created in this global context.

NoteCards allows us to examine the distinction between node-based and overview-based access to the hypertext since it provides users with both modes of interaction. Users generally regarded access using the graph-style overview (referred to as the browser) as clumsy and slow, and did not use it if they were engaged primarily in information management tasks. On the other hand, users faced with large structuring tasks frequently chose to work from the browser, in spite of its drawbacks. They would maintain browsers across sessions, and use them as a context-setting backdrop for their work. These browsers functioned as accelerators for accessing and referring to existing structures (see page 839 in [Hala88]). NoteCards users also developed strategies for using browsers to group nodes by means of spatial layout, especially in the very earliest brainstorming stages of a task (see page 96 in [Trig87]).

Because of the effectiveness of the overview in gIBIS and our experiences with the NoteCards browser, we decided to center Aquanet interactions around a graphical view of the knowledge structure.

2.2 Complex relations as a richer linking model

A second important artifact of emphasizing the information management and presentation aspects of hypertext is that links are designed primarily as navigational aids. Even though in systems like NoteCards and gIBIS, links are labelled or even typed, it is difficult to build coherent composite structures using links alone. The legal case cluster card described on page 843 of [Hala88] is a representative example. In this example, a NoteCards user attempted to represent the structure of a complex legal case, but failed because he could not adequately represent its overall structure using a simple collection of nodes and links.

We experienced analogous problems of reference and scope in representing Toulmin structures in NoteCards. As a result, we were forced to create extra mechanisms (specifically, a Toulmin card) to capture the semantics of the Toulmin relation [Mars87]. For similar reasons, the IDE developers were forced to build a structure library [Jord89] to facilitate their representation development. Even though in both of these cases the extra mechanism aided the task at hand, there were still significant limitations to its generality. For example, after structures were created through these mechanisms they were still maintained within the node-link model. Problems like the inability to ripple structural changes through all instances were endemic to both applications. Thus developing a richer linking model to express complex relations is an important goal of Aquanet.

3.0 The Aquanet Task: Collaborative knowledge structuring

Looking more closely at knowledge structuring tasks, we can distinguish between two kinds of activities. First, people *use* knowledge structures to organize and categorize content. Second, over the course of a task people *develop* knowledge structuring schemes; these schemes evolve through negotiation and use. In practice, the two activities aren't that easy to pull apart, but they each suggest a different set of use situations and user requirements.

Our own experiences using Toulmin structures as a descriptive way of organizing the content of reasoned discourse [Newm91] provide some interesting insight into how knowledge structures evolve as they are used collaboratively. Originally, we chose a fixed knowledge structuring scheme, Toulmin's model of argument [Toul58] (see Figure 1 on page 2), as the primary vehicle for performing a series of analyses. In the course of the analyses, we discovered that not only was

Toulmin's model insufficiently prescriptive, but it also failed to cover and highlight all of the phenomena of interest to us. To resolve the aspects of the model that were insufficiently prescriptive, we found ourselves negotiating what kinds of statements could be used as each of Toulmin's micro-argument elements (for example, we had to decide whether a datum could be a very general statement, or whether very general statements were always warrants) and how Toulmin structures could be hooked together (for example, we had to decide whether it was possible to use the backing of one argument as the claim in another). To cover the parts of the argument that were difficult to "Toulminize," we had to create new kinds of structures. Thus we found it necessary to both *constrain* and *extend* our original knowledge structuring scheme.

As the example above illustrates, knowledge structuring tasks are frequently collaborative; they can involve more than one person, and they usually take place over an extended period of time. The kind of knowledge structuring that we've observed is semi-synchronous rather than the tightly-coordinated synchronous collaboration that takes place in meetings. Much of the interaction among collaborators takes place through actions on the knowledge structures as well as through meta-comments attached to the knowledge structure.

The design and development of Aquanet was driven by these and other experiences using NoteCards for knowledge structuring tasks. Out of these experiences, we have derived the following requirements:

(1) To define a knowledge structuring scheme, a user must be able to specify what its elements are and how they are interconnected. For example, in an IBIS model, there are Issues, Positions, and Arguments, and the Arguments can support Positions, but they cannot respond to Issues.

(2) To develop a knowledge structure, a user must be able to modify and extend the knowledge structuring scheme as her understanding of the task changes. As we have suggested by our example, the evolution of structuring schemes is an unavoidable side-effect of using them.

(3) To build or use a knowledge structure in the collaborative settings described above, users must be able to see - with a reasonable delay - what other users have done. We call the kind of updating required for this style of interaction WYSIWID: What You See Is What I Did.

(4) To display multiple views onto a single knowledge structure, a user must be able to specify alternate graphic renderings of the same structure. For example, a user might want to see arguments about design options and criteria as a matrix at the same time as the information is shown as a dependency tree (see MacLean et al.'s QOC [MacL91] or Lee's DRL [Lee90]).

(5) To use combinations of methodologies in a single task (as in Streitz's activity spaces [Stre89]), a user must be able to compose knowledge structuring schemes. For example, a task that requires both issue-structuring and argumentation might combine an IBIS Issue-Position-Argument model and a Toulmin Data-Claim-Warrant argument model.

(6) To develop a knowledge structure collaboratively, users must be able to negotiate about its contents; they must be able to talk *about* the knowledge structure as well as *through* the knowledge structure (see also Conklin and Begeman's account of "going meta" in [Conk88]).

In designing Aquanet, we tried to address all six of these requirements. While the current implementation of the tool falls short of fully meeting all of them, it is our long term research goal to fully support the requirements of knowledge structuring tasks.

4.0 Knowledge Structuring Concepts in Aquanet

Aquanet is designed to support users in creating, storing, editing, and browsing large graphical knowledge structures.

4.1 The Aquanet data model: basic objects and relations

Knowledge structures in Aquanet are constructed according to a data model that merges the standard hypertext data model (e.g., see [Hala90]) with a fairly standard frame-based data model (e.g., see [Bobr77]). In line with its hypertext roots, Aquanet makes a strong distinction between data-containing objects, typically called "nodes" in the hypertext model, and relational objects, typically called "links" in the hypertext model. The 'nodes' in Aquanet are called *basic objects*, the 'links' are called *relations*. In line with its knowledge representation roots, Aquanet objects (both basic objects and relations) are typed, structured, frame-like entities.

Every Aquanet object is made up of an unordered set of named slots. Every slot has a value (or several values in the case of multi-valued slots). The value is restricted to be data of a given type (see discussion of Aquanet types below).

The distinction between basic objects and relations lies in the nature of the allowable slot values. In basic objects, all slots values are restricted to be primitive datatypes (e.g., text, images, numbers, strings, dates, etc.). Thus basic objects are analogous to standard hypertext nodes (e.g., cards in NoteCards, frames in KMS, documents in Intermedia, etc.) except that instead of having a single content they have a set of named contents.

In contrast to basic objects, the slots in relations may be entity-valued, i.e., the value is some other object (or several objects for multi-valued slots). Thus Aquanet relations are analogous to hypertext links in that they are structural elements that serve to connect other entities in the structure. In hypertext terms, relations are "node-to-node" n-ary links that can be anchored to either nodes (basic objects) or other links (relations). Unlike links, however, relations have what amounts to named and typed endpoints.

Every Aquanet object is an instance of some type. The definition of a type specifies the slots that make up objects of that type, the restrictions on the type(s) of the objects that can fill each of these slots, and the graphical appearance of the object (see Section 4.3 below). Aquanet type definitions are organized into a multiple inheritance hierarchy. Objects of a given type include not only the slots defined in their type but also the slots that they inherit from their supertype(s). The rules of inheritance in the Aquanet type hierarchy are taken directly from the CommonLisp Object System specification [Stee90].

4.2 Building knowledge structures

Complex knowledge structures are built in Aquanet using two basic mechanisms for composing relations: *inclusion* and *chaining*. In inclusion, one relation is included as the value of a slot in some other relation. Figure 2C (page 6) is an example of inclusion composition. In chaining, two or more relations are connected because some Aquanet object fills a slot - though not necessarily the same slot - in all of the relations. Figure 2D (page 6) illustrates chaining composition.

4.3 Graphic appearances

Both the hypertext and frame data models focus on structure *per se* and largely ignore the issue of

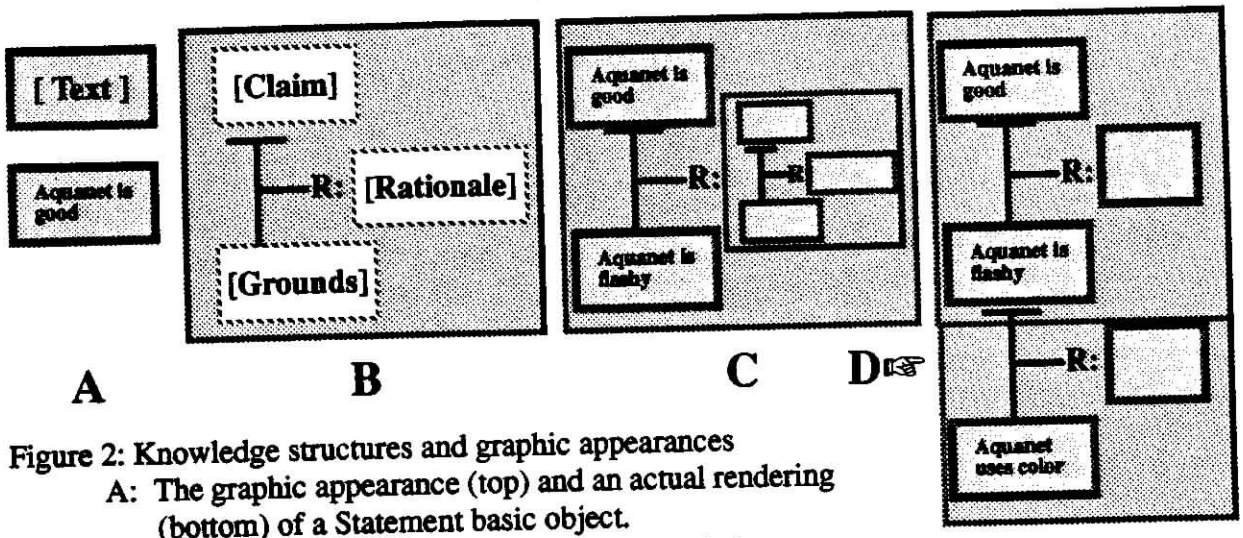


Figure 2: Knowledge structures and graphic appearances

A: The graphic appearance (top) and an actual rendering (bottom) of a Statement basic object.

B: The graphic appearance of an Argument relation.

C: An actual rendering of an Argument relation where two slots are filled by Statements and the third is filled by another Argument relation. This is an example of inclusion composition.

D: A rendering of two Argument relations that share a Statement object ("Aquanet is flashy"). This is an example of chaining composition.

how that structure is to be graphically displayed to the user. In contrast, the appearance of the knowledge structure is a critical component of the Aquanet data model. Specifically, every type definition includes information about the type's *graphic appearance.*

An Aquanet object's graphic appearance specifies exactly what the object and its slots should look like on the display. When rendering a knowledge structure, Aquanet reserves a rectangular region of the display for each object. The object's graphic appearance determines what is drawn into this rectangular region. Figure 2A (top) shows an example of the graphic appearance of a representative basic object (a Statement). Figure 2B shows a representative relation (an Argument). Statements have one important slot called Text. Arguments have three slots - the Claim, the Grounds, and the Rationale - each of which can be filled by a either a Statement or another Argument.

Graphical appearances can contain two types of items: *graphic elements* and *slot values.* The graphic elements are items such as lines, circles, squares, text labels, background colors, etc. that are drawn directly onto the display (scaled appropriately to fit into the allotted region). In Figure 2B, the vertical and horizontal black lines and the characters "R:" are examples of graphic elements.

Slot value items reserve an area on the display into which the value of a named slot will be rendered. For primitive valued slots, the value of the named slot is printed in this area (see the bottom illustration in Figure 2A). For entity-valued slots, the graphic appearance of the slot's value is recursively rendered (after the necessary scaling) into the reserved area. In Figure 2B, the three dashed boxes containing the bracketed slot names are slot value items. Figure 2C shows the rendering of an instantiated Argument relation. In this instantiation, two of the Argument's slots are filled with Statement basic objects and one is filled with another Argument relation.

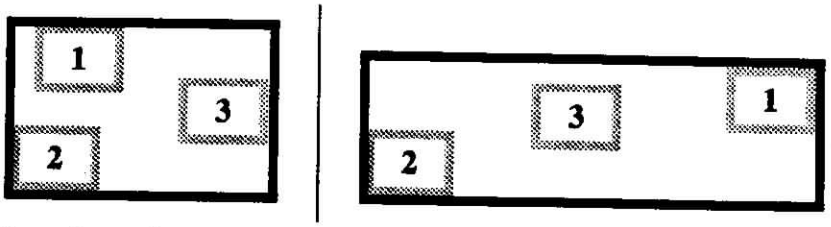


Figure 3: The rendering of an adjustable relation (black border) containing three basic objects (gray borders), both before (left side) and after (right side) the user moves the "1" basic

The graphical appearance mechanism just described is most appropriate for relations in which the layout of the relation follows a specifiable convention. Many relations have no such convention. For these cases, Aquanet allows the user to arbitrarily position objects on the display. The region occupied by the relation that contains these objects is then adjusted to be the bounding box just big enough to hold all of the contained objects. Figure 3 illustrates the operation of these adjustable relations. Such relations are used to construct standard network diagrams such as the browsers found in NoteCards and gIBIS.

Composing a 2-dimensional graphical appearance for a complex knowledge structure out of the graphical appearances of its component objects raises many challenging design issues. To avoid some of the very difficult layout mechanisms that would be required for a general solution, we chose to employ two simplifying mechanisms in our first implementation. First, Aquanet structures are arranged in a $2^1/_2$ dimensional space which is then rendered onto the 2 dimensional space of the display screen. Where the graphical appearances of objects overlap, one of the objects is stacked on top of the others, thus partially or fully obscuring them. The user can manipulate the stacking order of Aquanet objects to ensure that the desired object is displayed unobscured.

The second simplifying mechanism involves the creation of multiple views of a single Aquanet object (called *virtual copies* in the Aquanet interface) that can be placed at disparate locations on the display. Using virtual copies, the layout of a complex structure is simplified because the structure can be split into pieces that can be arbitrarily placed on the display without the constraint that logically connected relations (i.e., relations that share a common included object) be spatially co-located on the display.

4.4 Aquanet schemas

One of our major goals in designing Aquanet was to provide users with the ability to customize knowledge structures for their specific task. Aquanet accomplishes this goal through the use of *schemas*. Every Aquanet session is controlled by a schema that defines a set of allowable basic object and relation types. Since the basic object and relation types specify their slots and constrain the slots' values, the schema defines the nature and organization of the knowledge structure that the user can construct. For example, one can easily design an Toulmin schema that allows the user to create only one type of relation, the Toulmin relation, whose five slots (Datum, Claim, Warrant, Backing and Rebuttal) can only be filled by a Statement basic object. On the other hand, one could as easily design an IBIS schema that allows the user to create 3 types of basic objects (Issues, Positions, and Arguments) and connect them using the 9 types of relations (with appropriate type restrictions) that are described in [Conk88].

Aquanet includes a mechanism for composing schemas. Specifically, a schema can include by ref-

erence any other schema. The types in the included schema are added to the list of types in the including schema. The design of a more sophisticated composition mechanism that would, for example, provide for subtyping of schemas is a topic we are currently exploring.

The Aquanet schema language is somewhat limited in expressiveness. In particular, the schema determines the knowledge structure only on a local level. There is no way in Aquanet to express multi-object or global restrictions on the organization of the knowledge structure. For example, one cannot ensure that only one instance of a certain type exists in a knowledge structure, nor can one state that some object should not be related to itself through a series of connecting relations. A richer schema language is another area we are actively investigating.

Aquanet includes both a type and a schema editor. With the type editor, the user can easily access all of the properties of a type including its list of supertypes, its graphic appearance, its slots, and the slot value restrictions. With the schema editor, the user can add and remove types in a schema.

Allowing the user to edit types and schemas brings up a host of difficult issues about how to reconcile existing objects with schema changes. Although these issues of schema evolution are a critical focus for our work in the long term, we chose to simplify our initial implementation by restricting the possible changes that users can make when editing types and schemas. In particular, users can add and delete types to/from a schema and they can add/delete slots in a type. They can also arbitrarily change the graphical appearance of a type. Few other type or schema modifications are currently possible. For example, changing the type restriction on a slot value is not generally allowed. Since they are clearly at odds with our goal of supporting for schema evolution during knowledge structuring tasks, such restrictions on schema editing will be removed in future releases of Aquanet.

4.5 Aquanet discussions

Aquanet stores individual knowledge structures as well as schema information in a central, shared database. Individual knowledge structures are known as *discussions*, following the terminology used in gIBIS. Each discussion uses a single schema (but many discussions may share a single schema). At any given time, each instance of the Aquanet tool can only have a single open discussion. Objects contained in one discussion cannot "reference" (i.e., use as a slot value) objects in another discussion. Discussions are also the level at which most access control is enforced.

5.0 A Guided Tour of Aquanet

Users create and browse a graphical knowledge structure through the multi-paned Aquanet window shown in Figure 4. The knowledge structure shown uses the Argument schema described in Figure 2 (page 6). The pane on the left side of the window is a scrollable display onto the full structure. The top right pane displays a filtered list of objects in the network. This pane will be used to contain other kinds of user-specifiable views in future implementations. Objects can be selected using either of these two panes. The lower right pane displays the content - the primitive slots and their values - of the selected object. Objects displayed in this pane can be checked out from the database for editing.

Users can extend a structure in two different ways: they can use an existing object in a new role, thus creating a new relation, or they can create a new relation and fill its slots with new or existing objects. Figure 5a shows an example of how a structure grows by using an existing object, the Statement "Use of Xerox's network should be optimized," which is already Grounds for one argument, as the Conclusion for new argument. Figure 5b shows the results of this operation.

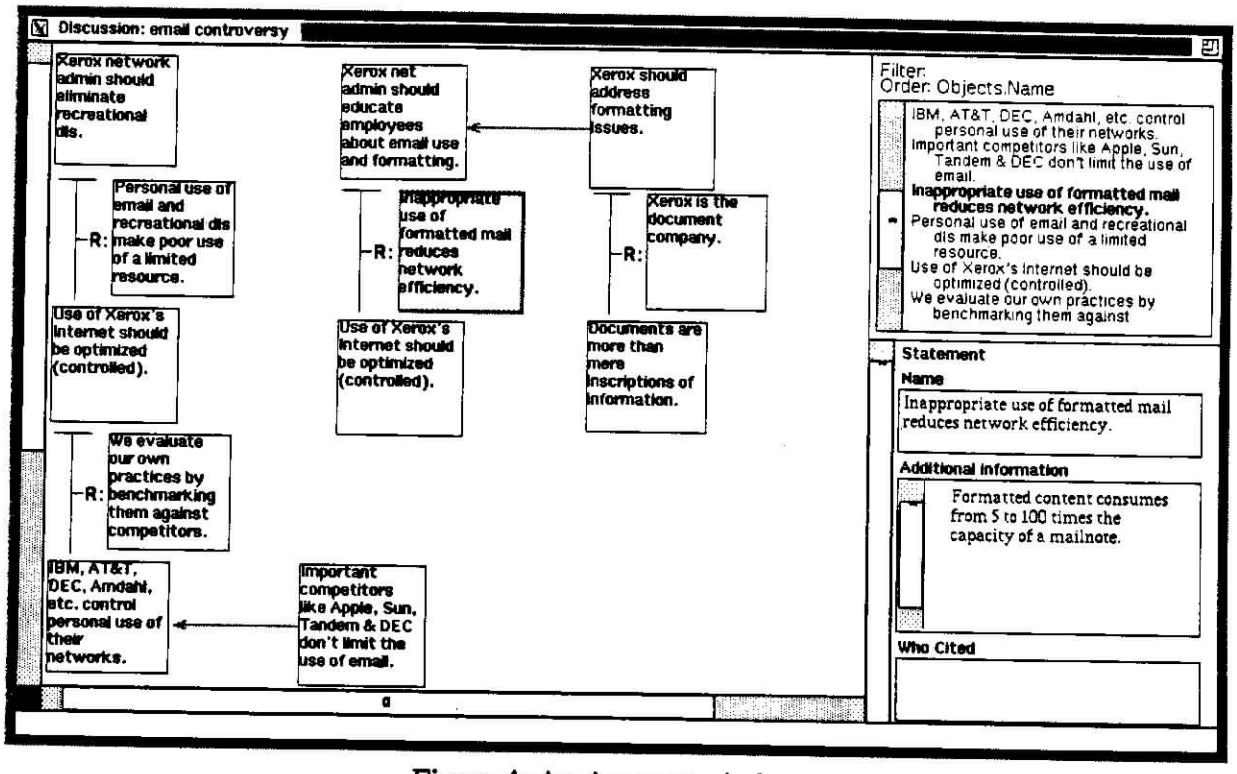
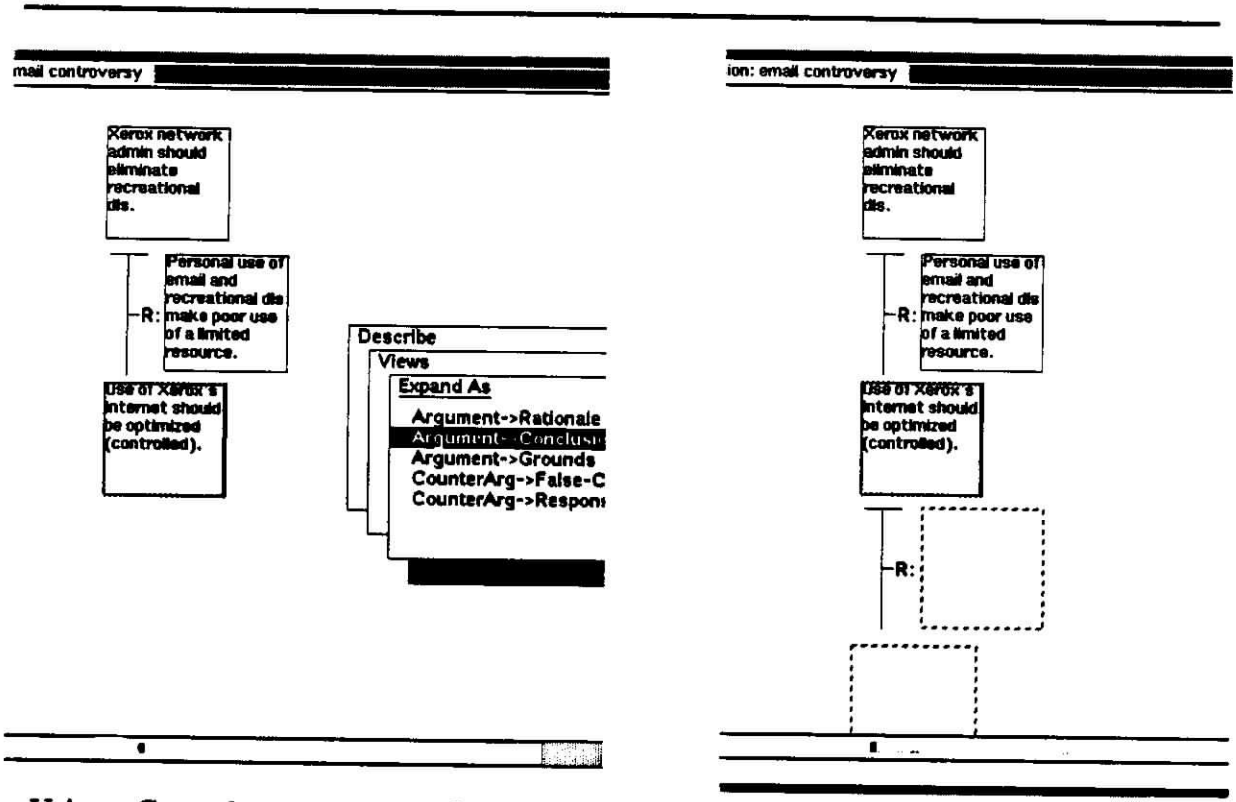


Figure 4: An Aquanet window.



a. Using a Grounds statement as a Conclusion

b. The extended network

Figure 5: Using an object in a new role

Users develop and modify schemas with a schema editor as shown in Figure 6. The types that are included in the schema may be selected from the list on the left side of the window.

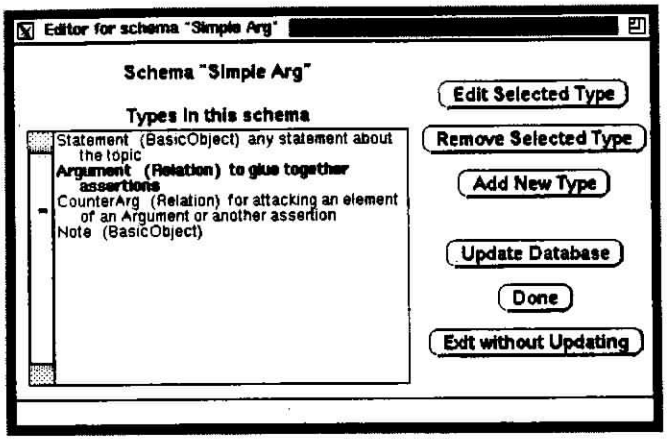


Figure 6: The schema editor. The Argument relation has been selected for editing.

Users edit types with a separate type editor. The type editor allows users to name a type, list its supertypes, define its slots, and specify its graphic appearance. Figure 7 shows the type editor invoked on the Argument relation. It has no supertypes. It contains three entity-valued slots, Grounds, Conclusion, and Rationale. Its graphic appearance is shown in the editing pane on the right hand side of the window. In Figure 7, one of the lines has been selected; a user is changing its color to red and its width to 2.

6.0 Implementation Notes

Aquanet runs on Unix workstations and can be used from any workstation or personal computer that supports X Windows. A color monitor is useful but not required. Aquanet is written in single inheritance object extension to "C" and built on the Andrew Toolkit[Pala88]. A Sybase database server is used to store the knowledge structures and the user-defined schemas. Sybase also manages the concurrency control necessary to support multi-user collaboration.

The Aquanet object-oriented model is mapped onto the Sybase relational database. Each type is stored as a separate database table. Another set of tables stores the schema information.

To support semi-synchronous collaboration, the central database server mediates access and changes to each discussion. Before an object is changed, it is locked in the database and its latest state is recached locally. When the object is unlocked, the edit is noted in a change log. Each session periodically polls this log to see which elements have been changed. Any new or changed elements are recached, resulting in an up-to-date view of the discussion.The default polling interval is 30 seconds.

The implementation uses a data / view architecture; the discussion is stored in a data object and multiple views are created to display the knowledge structure. A user, by interacting with one of the views, edits the knowledge structure which broadcasts the change to all of its views. Each view responds, updating its appearance to match the new state of the discussion. This facilitates viewing and interacting with different representations of the discussion while preserving consistency across these views.

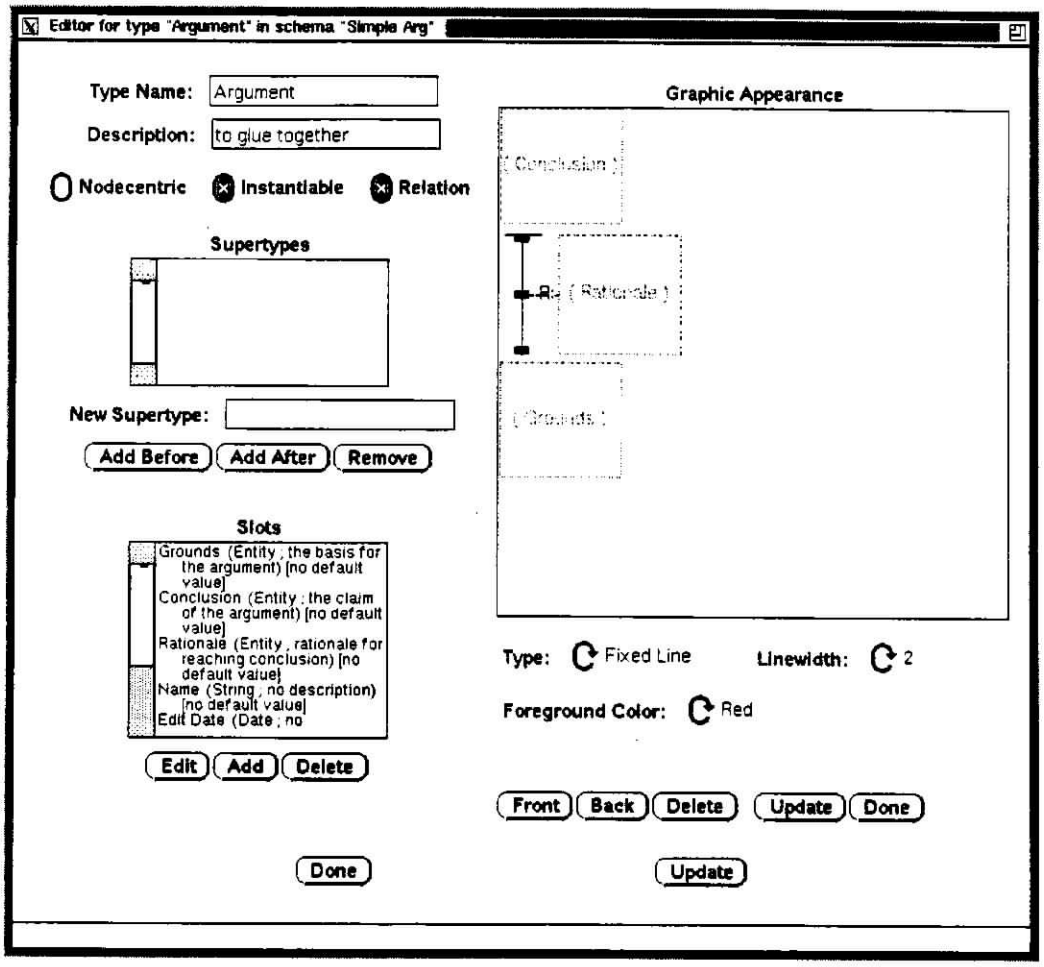


Figure 7: Editing a type

7.0 Early Experiences with Aquanet

We have just released the aerosol (alpha) version of Aquanet. Our early use experiences are drawn both from our own collaborations and from the experiences of some tolerant pre-alpha users. Already we have accumulated a diverse collection of schemas that reflect the varying representational needs of the initial set of tasks. We have also noticed some important patterns in these early experiences and have noted some additional requirements central to using the tool for its intended purposes.

To date, Aquanet discussions have been initiated in service of analytic tasks (one is a competitive analysis of current machine translation efforts), group design efforts (for example, designing a new drawing editor), organizing an existing collection of structured information (usage notes for a dictionary), and more general collaborative work (we use it to keep track of Aquanet bugs and feature requests and we've used it to organize this paper). While many of these discussions are still in early phases of organization, they have already caused us to reflect on some general issues. First, we have noted that people invent ways of creating lightweight structure to lessen the problems of premature organization. Second, we have looked at the kinds of schemas people have come up with, how they differ, and how they share certain characteristics. Finally, we have recog-

nized the difficulties with bringing information into the tool and defining output when structuring must result in a linear document.

7.1 Using spatial organization

As noted in [Stef87], [Trig87], and elsewhere, spatial organization of nodes is an important lightweight way of creating structure. In Aquanet, users have created representational types like labels that help them partition and differentiate a large space. Furthermore, when simple grouping relations like the one shown in Figure 3 (page 7) have been available in addition to labels, users have shown a preference for labels; they just manipulate layout to indicate relationships among objects.

Some discussions use the spatial characteristics of a layout to partition a task among members of a work group. For example, in writing this paper, after initial brainstorming about topics, we each chose lists of nodes under a given topic, and created our own "work areas." Similarly, in our discussion about the system itself (a discussion of the current bugs and desired features), certain areas of the main view became the "property" of a particular author, while others tended to be group efforts.

Spatial layout also provides some useful cues for readers navigating a large discussion, especially since the current implementation of Aquanet provides no birds-eye view of the entire structure. For example, the usage notes discussion uses spatial/alphabetic organization of its entries to aid the author and any potential readers in scrolling through the large space.

Given the prevalence of these spatial organization and layout strategies, it is clear that future versions of Aquanet should provide explicit support for them.

7.2 Supporting common representational underpinnings

Our early experiences with Aquanet have shown us that schemas will differ radically given different tasks and different people involved in a discussion. Most of the schemas to date are highly specific to the content of the discussion. For example, for our discussion of Aquanet bugs and features, we have developed a schema that includes types like "Bugs," "Features," and "Architectural Changes," and relations like "Fixes" and "Clumps." Other schemas are more general, like the one expressing the Toulmin relation that we've used in our earlier examples; the types that it provides do not necessarily correspond to specific characteristics of the task at hand. Our intuition is that the more general schemas are either more difficult to use - it is harder to get people to agree on a content-representation mapping - or that they aren't as powerful in structuring content since useful distinctions aren't brought out.

Although we've discovered that schemas may differ radically, we've also found that some representational needs don't vary that much across applications. One of the first types that we created was modeled after a post-it note. This type is displayed as a bright yellow rectangular background with the contents of a note shown in the foreground in a large font. It proved to be so useful, especially for talking about the schema and for quick annotations, that we included it in many other schemas. We also noted several useful representational primitives like groups, ordered lists, and binary links; these types recur in variant forms in many schemas.

One way we have considered providing users with sets of standard types like lists, groups, outlines, labels, and post-its is as a schema library. Schema developers could then specialize these generally useful constructs for their applications, or include meta-discussion objects like post-its

and labels in any discussion.

7.3 Information import and export

Users often come to knowledge structuring tasks with a collection of existing material and some preliminary ideas about how this material should be structured. For example, in the machine translation analysis, we have some on-line sources pulled from USENET and COMLINE; these on-line sources have well-defined internal structure and could easily be automatically integrated into the task's knowledge structure. Automatic import of external information would not only relieve the tedium of entering such information by hand, it would also promote consistent encoding of the information. We are currently designing general mechanisms for importing information into Aquanet.

The converse of the import issue is the externalization of knowledge structures from Aquanet into forms usable by other tools. Because many early Aquanet users requested such a facility, we have implemented a simple report generation program. A more sophisticated externalization capability is also being developed.

8.0 Dangling Links

Since Aquanet is just now ready for general use, our major focus in the near future will be on building a user community. The feedback we receive from people using the tool in real tasks will help direct its future evolution. Based on early observations, we intend to pursue the following research themes:

(1) Extend methods of viewing and interacting with knowledge structures. Specifically, we intend to allow users to define schema-specific viewers. We also will design a generalized network description language.

(2) Provide better support for the schema evolution process and the exploration of alternative structures. This includes a versioning mechanism for both schemas and discussions.

(3) Provide mechanisms for integrating Aquanet with the computing environment. This includes a programmer's interface.

(4) Provide a mechanism for knowledge structure computations, including associating behaviors with types, schemas, and objects. This mechanism would support for dynamic objects, computed role values, and intelligent discussion "agents" for example.

Acknowledgements

We would like to thank Tom Moran, Susan Newman, Jintae Lee, and Norbert Streitz for their helpful comments during the development of Aquanet.

References

[Appl87] Apple Computer, Inc. Hypercard User's Guide

[Aksc88] Akscyn, R., McCraken, D., & Yoder, E. KMS: A Distributed Hypermedia System for Managing Knowledge in Organizations. CACM 31, 7 (July 1988), 820-835

[Bobr77] Bobrow, D.G. & Winograd, T. An Overview of KRL,A Knowledge Representation Language. *Cognitive Science*, 1(1), 1977, 3-46.

[Brow85] Brown, J.S., & Newman, S.E. Issues in Cognitive and Social Ergonomics: From Our House to Bauhaus. *Human-Computer Interaction*, 1985, 1(4), 359-391.

[Brow87] Brown, P.J., Turning Ideas into Products: The Guide System. Hypertext '87 Papers, Chapel Hill, North Carolina, November 13-15, 1987

[Brun88] Bruns, G. Germ: A Metasystem for Browsing and Editing. MCC Technical Report STP-122-88

[Bush45] Bush, V. As We May Think. Atlantic Monthly, August 1945, 101-108

[Conk88] Conklin, J. and Begeman, M.L., "gIBIS: A Hypertext Tool for Exploratory Policy Discussion," MCC Technical Report Number STP-082-88, Austin, Texas, 1988.

[Engl68] Engelbart, D.C., English, W.K. A Research Center for Augmenting Human Intellect. Proceedings of the 1968 Fall Joint Computer Conference, 33 Part 1, Montvale, N.J.: AFIPS Press, 1968, 395-410

[Garr85] Garrett, L.N., Smith, K.E., & Meyrowitz, N. Intermedia:Issues, Strategies, and Tactics in the Design of a Hypermedia Document System. CSCW '86 Proceedings, Austin, Texas, December 3-5, 1986

[Hala87] Halasz, F.G., Moran, T.P., & Trigg, R.H. NoteCards in a Nutshell. Proceedings of the ACM CHI + GI Conference, pp45-52, Toronto, 1987

[Hala88] Halasz, F.G., Reflections on NoteCards: Seven Issues for the Next Generation of Hypermedia Systems.CACM, 31, 7, (July 1988), 836-852.

[Hala90] Halasz, F.G. & Schwartz, M. The Dexter Hypertext Reference Model. Proceedings of the Hypertext Standardization Workshop, National Institute of Standards and Technology, January 16-18, 1990. Available as NIST Special Publication 500-178, March 1990.

[Jord89] Jordan, D.S., Russell, D.M., Jensen,A.M., & Rogers,R.A. Facilitating the Development of Represenations in Hypertext with IDE. Hypertext '89 Proceedings, Pittsburg, Pennsylvania, November 5-8, 1989

[Lee90] Lee, J. "SIBYL: A Qualitative Decision Management System," to appear in Winston, P. and S. Shellard (Eds.) Artificial Intelligence at MIT: Expanding Frontiers, Chapter 5, The MIT Press: Cambridge, MA, 1990.

[Mars87] Marshall, C.C., Exploring Representation Problems using Hypertext. Hypertext '87 Papers, Chapel Hill, North Carolina, November 13-15, 1987

[Mars89] Marshall, C.C. "Representing the Structure of a Legal Argument," Proceedings of the Second International Conerfence on AI and Law, Vancouver, British Columbia, June 14-16, 1989

[MacL91] MacLean, A., Bellotti, V.M.E., & Moran, T.P. Questions,Options, and Criteria: Elements of Design Space Analysis. Journal of Human-Computer Interaction. Vol 6.

[Pala88] Palay, A., et al. The Andrew Toolkit: An Overview. Proceedings of the USENIX Technical Conference, February 1988

[Newm91] Newman, S.E., Marshall, C.C., Pushing Toulmin Too Far: Learning From an Argument Representation Scheme. Xerox PARC Technical Report, 1991.

[Russ87] Russell, D.M., Moran, T.P., & Jordan, D.S. The Instructional Design Environment. In Intelligent Tutoring Systems: Lessons Learned. JPsotka, L.D. Massey, & S.A. Mutter (Eds). Lawerence Erlbaum Associates, Inc. Hillsdale, N.J. 1987

[Stee90] Steele, G.L. *Common Lisp: The Language (Second Edition)* Digital Press, Bedford, MA, 1990.

[Stef87] Stefik, M., Foster, G., Bobrow, D., Kahn, K., Lanning, S., and Suchman, L., "Beyond the Chalkboard: Computer Support for Collaboration and Problem Solving in Meetings," Communications of the ACM, Vol. 30, No. 1., January, 1987, pp. 32-47.

[Stre89] Streitz, N.A., Hannemann, J., and Thuring, M., "From Ideas and Arguments to Hyperdocuments: Travelling through Activity Spaces," Hypertext '89 Proceedings, Pittsburgh, PA November, 1989.

[Toul58] Toulmin, S., The Uses of Argument, Cambridge University Press, Cambridge, 1958.

[Trig87] Trigg, R.H., Irish, P.M. Hypertext Habitats: Experiences of Writers in NoteCards. Hypertext '87 Papers, Chapel Hill, North Carolina, November 13-15, 1987

[Yake90] Yakemovic,K.C.B., Conklin, E.J. Report on a Development Project Use of an Issue-based Information System. Proceedings of the Conference on Computer-Supported Cooperative Work, Los Angeles, California,, October 7-10, 1990

6. Known bugs and caveats

This document lists bugs and some caveats about features not yet implemented, network-related "effects," and other problems; it is the latest version of Section 6 of the User's Manual.

6.1 Features not yet implemented.

Aquanet has no undo. Aquanet has no undo for any of its operations. This is clearly a desirable feature for some operations like delete, but it has not been implemented yet.

Aquanet has no "abort session." Aquanet has no way of removing all your changes from a discussion. So if you enter a discussion, make some contributions, and then want to exit, nullifying whatever you've done during the session, you must delete all nodes you've created. Otherwise your contribution will remain in the Aquanet database.

The z dimension (node depth) does not get saved. Aquanet currently does not save the depth of nodes. Thus, if you've arranged a group of overlapping nodes by an artful series of Tops and Bottoms, your arrangement will not be preserved after you've exited the discussion.

Moving nodes does not initiate a lock. Because moving a node may initiate a ripple effect through a large interconnected network of other nodes, it is possible for several users to get into a tug-of-war. If this happens, let the other user move first, since the last move is the one that will be stored in the database. Performance considerations make node-locking on move a prohibitively expensive solution.

Zoom and Pan Out are not implemented. The Zoom and Pan Out options on the Discussion menu are not fully implemented yet. If either is used, you will find that scaling is erratic. However, it's easy to recover if you accidentally either Zoom in or Pan Out since each of the operations exactly reverses the other.

Relayout is not implemented. Relayout is not implemented.

6.2 Side effects.

Loss of database connection. Occasionally you will lose your connection with Sybase, either because the network is flaky or there are too many people trying to access Sybase at the same time, and Aquanet will temporarily freeze.

6.3 Bugs.

Menus are occasionally incomplete. If the menu that appears when you middle click in a window does not correspond to the documentation, just try it again. Sometimes you may have to click left to change the window focus. This is particularly true for the discussion menu.

Drag updates are slow. Sometimes dragging nodes around in the main view can be annoyingly slow. We expect to fix this problem in the bouffant release of the tool.

Relations sometimes don't display their slots in the proper position. Dragging relations a minor distance sometimes will cause slot contents to be "left behind." This is a display bug - the slot contents are intact and will often display properly if you move them again.

If you try to delete a relation you don't own, you will clear its slots. Although you can't delete a relation you don't own, if you try, the nodes filling its slots will be cleared (ie. freed) from the relation.

The schema editor's toggles, choosers, and buttons look strange. If the schema editor's toggles, choosers, and buttons don't look like they do in the figures in this user's guide, you may have a missing font. To get this font, change your font path by typing the following command into your shell window:

```
xset +fp /import/local/andrew/X11fonts
```

Bugs not on this list can be reported to AquanetSupport.parc@xerox.com.