FRAME-BASED COMPUTER NETWORK MONITORING

Lawrence A. Stabile
Prime Computer, Inc. *
500 Old Connecticut Path
Framingham, Massachusetts 01701

Abstract

This paper describes an application of FRL to the monitoring of a large computer network. The design utilizes frame models for the network domain and a general graphics system, and a method for coupling the two domains by analogy creation and maintenance. We also explore the use of heuristics to control the selection of viewable descriptions, at their various abstraction levels and sizes. An ultimate goal of the project is to provide a convenient means for developers to create new graphic descriptions of the domain, without worry as to size of the description, and allow the heuristics to decide how and when the descriptions should be placed on the screen.

Introduction

We present the design of a frame-based system for monitoring and displaying the properties of a large computer network. We focus on (1) the development of a frame model for the computer network domain, including relevant information regarding performance, operational status, and topology, (2) the design of a frame model for graphic displays, a general constraint-based structure of hierarchical regions that may be manipulated by the user or the network domain; it includes the notion of "intelligent pan and zoom", described further below, and (3) the use of analogy creation and maintenance to join these two domains to effect the desired interactive behavior and promote the modularity of state-based structures. This project uses a version of FRL [7,8], briefly described in a later section.

This work bears similarity to the I-Space project of Rieger et. al. [6], and to various constraint-based simulation-oriented systems, such as Thinglab [1], and Constraints [9]. A distinction between our project and I-Space is our explicit attack on the problem of the size of graphic description: a terminal screen is of limited resolution; it represents an ultimate constraint in graphic-system design. The author sees this problem as central to the control of large data sets; our solution is to use domain-independent and -dependent heuristics on a homomorphic correspondence between the application domain (here, computer networks) and the graphics domain. This manifests itself as a so-called generic analogy, which we describe below.

----

* Present address:
Apollo Computer, Inc.,
19 Alpha Road,
Chelmsford, Mass. 01824

Goals

We desire to construct a system that permits the centralized observation of all activities on a large network of computers (in our case, a token-based ring network of approximately 50 Prime 750 systems). Typically, the user of such a system will be an operator or system administrator who needs to view an entire field of machines at once.

The desired characteristics of the network monitor are:

(1) The continuous monitoring and abstraction of important system events, such as faults, illegal attempts to access, and configuration changes.

(2) The observation of system performance via abstractions such as Cartesian graphs, tables, and flow diagrams, to display individual and overall machine loads, available disk space, topology, and other properties.

(3) The presentation of the above items to the user in an intelligent fashion on a (preferably color) graphics display.

Design of the Network Monitor

With the above goals in mind, it was natural to choose a system like FRL for its implementation. Its active nature (via attached procedures) provides us with the ability to make inferences, enforce constraints, and propagate information. This gives us the ability to use the representations as "physical things" that change appropriately when kicked or poked, just as real "things" should.

In the following paragraphs, we will first desribe the overall structure of our system, the network domain model, then the graphics model, and last the analogical link constructed between the two.

Overall operation of the network monitor is as follows. We start with a large network of machines, in this case a token-based ring. Running on each machine is a process that periodically sends a package of raw data back to the master monitoring process. Here, the raw data is placed under the appropriate slots in the frame or frames that model that machine, or associated communication links (virtual circuits). Placement of this data trggers demons that compute desired features and abstractions, which in turn may trigger further computations.

When sufficient data has been computed, demons that recognize this fact pass the data to the corresponding graphics frame via the analogical link (should one exist). Should the user be observing an updated piece of data, this update will immediately appear on the screen. The user may choose, via keyboard and/or tablet, to observe some other data or to probe a given datum in more detail. The graphics system will then be "connected", via analogy, to that data, and observation proceeds from there.

The frame structure for the network domain starts with the object "ring-net", which contains as components sets of machines and communication links. These frames in turn have components, such as user sets and peripheral devices. In addition, the frames contain various features and demons to compute these features as data is added.

As an example of operation of the network frames, consider some parameters of an individual machine. The raw data from the measurement processes contains cpu-time, io-time, and page-fault-rate. The addition of any of these to their slots in a machine frame causes computation of a general "load" figure. Similarly, other performance parameters, network topology, peripheral status, and other data, trigger demons appropriate to the desired abstractions. The demon-inheritance mechanisms of FRL are very important here; for instance, general topology changes are handled high in the ako-structure for the net, whereas more specific computations reside at the low level. The demons which may be triggered by such data-changes thus range from entire recomputation of the graphics analogy (e.g., a result of a topology change), to simple changes of one text string to another.

The graphics frame system is a general-purpose graphics tool with which one can build several kinds of hierarchical structures. Regions are contained within regions down to a primitive level; these are the objects that directly drive the terminal at hand, and are typically lines or strings, although any program that directly drives the terminal may be interfaced to the system to act as a primitive object (e.g., a bar graph). The basic unit used to size and group the data is the rectangle; a "compute-size" demon calculates the enclosing rectangle of a set of rectangles, given their relative coordinates. Special demons for each kind of layout desired position the objects in the appropriate way, relative to the enclosing object. Size computations propagate from the bottom up, thus giving the layout-designer freedom from screen-size constraints. One can build objects from other objects, construct new layout-demons, or add to the set of primitives, without regard to size - choice of properly-sized objects is done heuristically, as described below. Current elements of the layout library include center-justified-stacks of rectangles, top-justified-strings (and variations on these), elliptical layouts, text strings of various font sizes, bar graphs, and individually pan/zoom-able (in the physical sense) Cartesian graphs.

With the network domain model and the graphics model so described, our task now is to link them together. We do this by creating and maintaining an _analogy_ between the two structures. In our system, this method forms a fundamental engineering structuring technique: state-based systems may be built _independently_, then coupled via the analogy mechanism. This is similar to using composition in a functional model, but here the state behavior of each system is naturally preserved and utilized.

An analogy is simply a correspondence between the components of two frame structures such that changes in the components of one domain are naturally reflected in the other. A slot name is chosen for the correspondence, such as "full-graphic-description" or "summary-graphic-description". The correspondence is currently produced by demons in the source domain that instantiate the target frames and connect the analogical link. Recursive domains are supported simply by requiring an analogy-creating demon to construct a target frame using components that are analogies of the components of the source frame.

Although many analogies are built as correspondences between individual frames, the truly interesting case occurs when we create a correspondence between an _individual_ and a _generic_ structure. We thus have a _homomorphism_ in which any object that may be instantiated in the target prototype validly represents the source object. The user, and the system-heuristics, may freely choose an object to instantiate depending on size constraints, desired level of abstraction, and other general or domain-specific information.

In our monitoring system, the individual is the network model and the generic structure is the set of possible graphic descriptions of that model. For example, a ring-net may have a description which is graphically depicted as an elliptical layout, with small objects for each machine-node that may indicate nothing more than "up", "down", "loaded", or "unused". One could thus fit this summary of many machines on a single screen. A more detailed description might include bar graphs of performance of each machine, but one could then see only a small number of them at a time. The variety of descriptions continues in this way; what the user sees is determined by desire, size, detail, and importance.

The resulting generic-graphics structure forms an AND/OR graph; any tree built from this graph is a valid description of the domain. To obtain a size-sorted set of descriptions for pan-zoom, we note that since size propagates upwards, we could simply enumerate the cross-product of all choices, size-sort them, and allow the user to choose a specific structure. This is, unfortunately, potentially explosive combinatorially. We thus use both graphics-domain and network-domain heuristics to determine the appropriate set of objects to build. Some of the heuristics are: (1) Allow the user (in the network domain) to attach intuitive information as to the relative size and importance of the graphic descriptions, (2) Use limited-depth search of the graphics AND/OR graph to approximate exhaustive enumeration, (3) Use general domain-properties, e.g., a table may be too large to fit on the screen and could be divided into "pages", (4) Use the "uniform level hypothesis" that many domains are naturally structured as parts decompositions that are of the same kind at each level (e.g., in digital circuits, we go from block diagram -> gates -> transistors -> holes and electrons), (5) Be able to correct any erroneous choices made above by suitable backtracking.

Implementation Status

Currently, a small version of the network monitor runs on five machines. The network-domain frames receive data over X.25 virtual circuits; performance data is currently the only data gathered.

The graphics system is complete, contains many primitive objects and layouts, and operates completely constraint-based. It has been tested on various small domains (including recursive ones) with success. Heuristics for pan and zoom currently consist of user-supplied size intuitions and a simple limited-depth size search. It is anticipated that the ultimate usefulness of the heuristics described earlier will not be ascertainable without using larger data sets. Backtracking is installed, and is currently user-controlled.

FRL is implemented as a set of primitives inside our interpretive Lisp system. The performance of this system will have to be improved for truly usable results, although it is acceptable on the current small model. Our version of FRL is much like the original [7,8], but we have not implemented some of the more esoteric functions, and have added a few extensions: *if-added, *if-removed, etc., are properties attached to the <u>frame definition</u> of <u>slots</u>. These are useful for bookkeeping kinds of slots that may have a variable number of occurrences in a frame (e.g., component-slots), for installing inverses of slots, and (eventually) for controlling analogies in a constraint-based way. Also, we have installed a simple queuing mechanism to suspend attached procedures (via closures) when their needed data is unavailable. Similar to the <u>monitors</u> of Fikes [2], this is useful for non-constraint-based inferences, and for resolving "forward references" when loading data from a text file.

The network communications facility is embedded in Lisp as a few simple functions to start processes and to send and receive messages. The underlying mechanism for this is a set of inter-process communication primitives (see [3] for details). For an overview of Lisp-based network communication techniques, see Model [4].

Problems and Directions for Future Research

Localization of naming scopes within a FRL domain would be of great benefit, allowing independent designers to combine frame systems without name-conflict worry. We are considering extensions in this direction. In addition, a troublesome aspect of FRL is the lack of clear distinction between knowledge and meta-knowledge. This crops up in spots where we wish to build, say, frames whose constraints control the way in which other frames and their constraints are built.

We also wish to apply our system to the control of networks, as well as their monitoring, and to utilize some of the ideas from expert systems and pattern-matching in the monitoring and control process, a goal shared by the I-Space project [6].

Despite the problems mentioned above, the author has found the object-based method to be of great use in structuring data and their interactions, with particular freedom from sequential control.

References

[1] Borning, A., "Thinglab - An Object-Oriented System for Building Simulations using Constraints", ICJAI-5, MIT, Cambridge, Mass., August, 1977.

[2] Fikes, R.E., "Odyssey: A Knowledge-Based Assistant", Artificial Intelligence, 16-3, July, 1981.

[3] Gertner, I., "A Report on Process Management in a Guest Distributed Operating System", Third Intl. Conf. on Distributed Computing Systems, October, 1982, Ft. Lauderdale, Florida.

[4] Model, M.L., "Multiprocessing via Intercommunicating Lisp Systems", Proc., 1980 Lisp Conference, Stanford, California.

[5] Rieger, C., and C. Stanfill, "Real-time Causal Monitors for Complex Physical Sites", Proc., NCAI-1, Stanford, California, 1980.

[6] Rieger, C., R. Wood, and E. Allen, "Large Human-Machine Information Spaces", Proc., IJCAI-81, Vancouver, BC, 1981.

[7] Roberts, R.B., and I.P. Goldstein, "The FRL Primer", MIT-AI Lab Memo 408, 1977.

[8] Roberts, R.B., and I.P. Goldstein, "The FRL Manual", MIT-AI Lab Memo 409, 1977.

[9] Sussman, G. J., and G.L. Steele, Jr., "Constraints-A Language for Expressing Almost-Hierarchical Descriptions", Artificial Intelligence 14-1, August, 1980.

[10] Winston, P.H., "Learning and Reasoning by Analogy", CACM, 23-12, December, 1980.