A Local Area Computer Network Expert System Framework

Robert Dominy/Code 522.1
NASA Goddard Space Flight Center
Greenbelt, MD 20771

Abstract

Over the past year I have been developing an expert system called LANES designed to detect and isolate faults in the Goddard-wide Hybrid Local Area Computer Network (LACN). As a result, the need for developing a more generic LACN fault isolation expert system has become apparent. This paper explores an object oriented approach to create a set of generic classes, objects, rules, and methods that would be necessary to meet this need. The object classes provide a convenient mechanism for separating high level information from low level network specific information. This approach yields a framework which can be applied to different network configurations and be easily expanded to meet new needs.

Introduction

A hybrid local area computer network (LACN) connects a variety of computers, ranging from mainframes to micros, throughout several buildings of the Goddard Space Flight Center. The network is based primarily on an Ethernet bus topology (Figure 1) but a star configuration using the Cable TV (CATV) wiring and Applitek equipment allows networks in different buildings to be interconnected.

As this network grows, it becomes more and more difficult to detect and isolate problems. Currently operators do a small amount of network checking manually, but to do a thorough check of the entire network is tedious because of the large number of items involved. As a result, most fault isolation takes place after a user has discovered and reported a problem.

The Local Area Network Expert System (LANES) began in 1985 as an effort to automate the process of fault detection and isolation. Development work is being performed on a Symbolics 3645 LISP machine using an expert

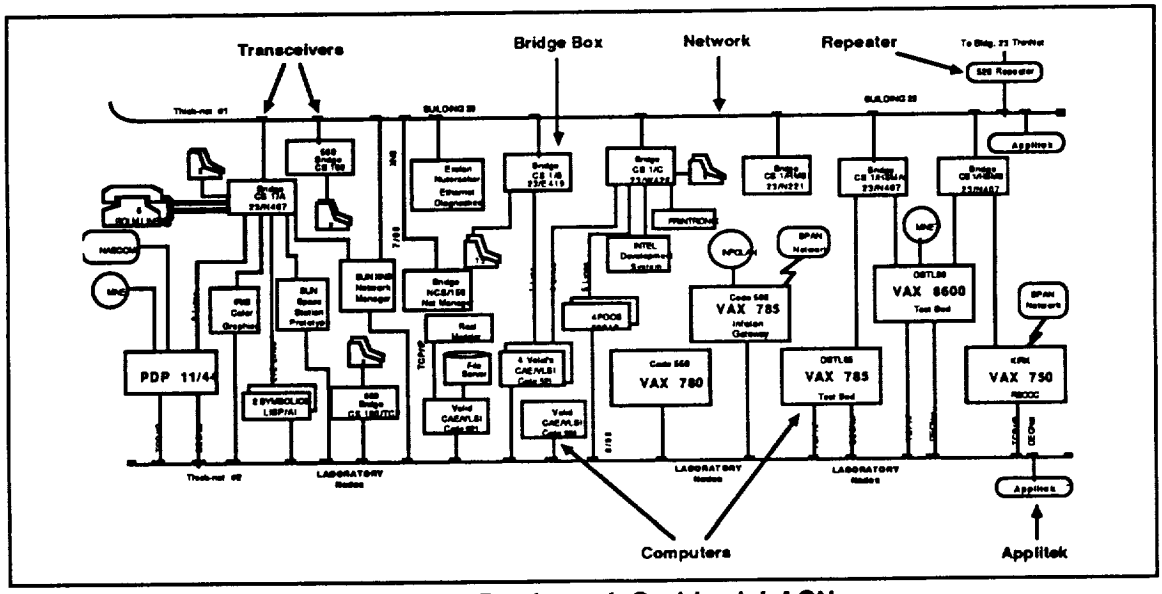

Figure 1 -- Portion of Goddard LACN

system development tool, the Automated Reasoning Tool (ART) from Inference Corp. An operationally useful prototype has been completed and is available for demonstration and use.

The LANES screen is depicted in Figure 2. Each network is depicted graphically and a global map of Goddard Space Flight Center that shows connections between buildings (Figure 3) can be reached by clicking on the window connection icon (the upward arrow) labeled Map. To tell the system to begin checking the entire network a user clicks on the START button by positioning the mouse cursor over the START button and pressing the left mouse button. The button changes to STOP. The expert system heuristic rules then attempt to detect and isolate problems in the network. Results are displayed with text and graphics. For example, in Figure 2 the Bridge box CS1-D cannot be reached (an X is placed in the box) and the Bridge box CS1-B has a high statistical CRC (cycle redundancy check) error count (a question mark is placed in the box).

LANES checks each device on each network, repeatedly cycling through all the networks, until the user clicks on STOP. As each network is checked, the window being viewed will automatically be changed if the current network is on a different screen and the Switch Screens button is set. At any time the user can edit the network design using NetEdit, an interactive graphical network editor. Devices can be added, moved, modified, or removed. The changes are reflected immediately by the expert system and can be used temporarily or saved to a permanent

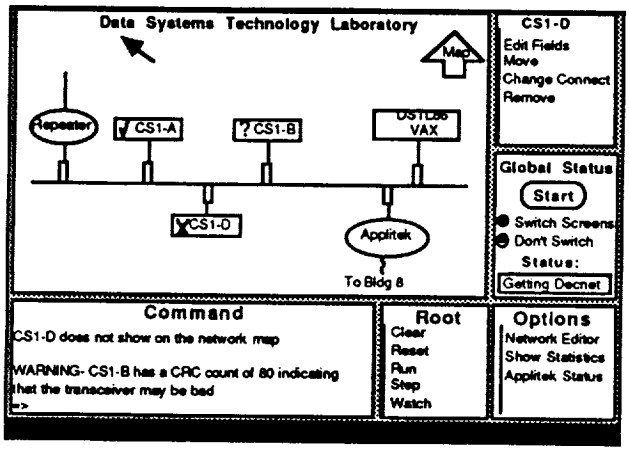

Figure 2 --LANES Screen

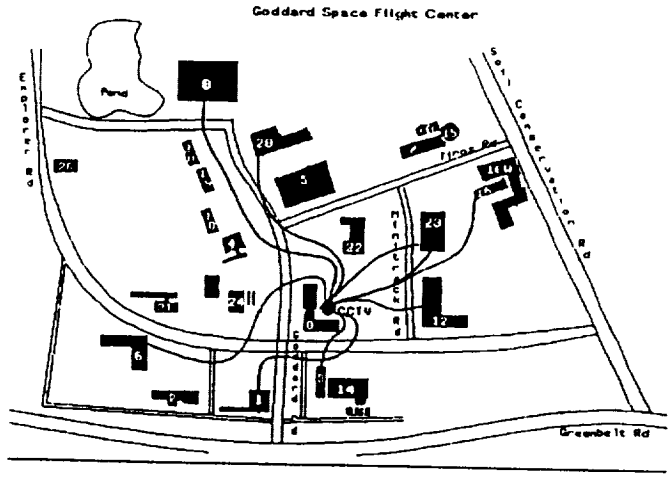

Figure 3 -- Map of Goddard

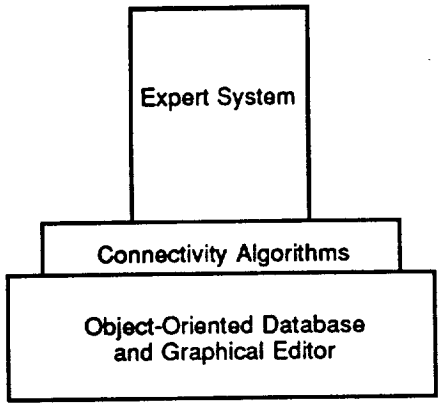

Figure 4 -- Framework Layers

file.

The current version of LANES checks networks located in five different buildings. It gets the up/down status and health statistics of seventeen bridge boxes, gets the up/down status of ten VAX computers, and derives the status of five Applitek devices.

Object-Oriented Framework

Demonstrating LANES has generated interest within Goddard and at other NASA centers. However, because LANES has been designed specifically for the Goddard LACN much of the code that was developed would have little application to other LACNs. In addition, a number of functions and rules within LANES are replicated for similar object types. For example two rules which get the status of an object may only differ by the function which they invoke. This not only duplicates code and effort, but also makes global changes more difficult to make. Improved knowledge abstraction with increased modularity and ease of modification are being added to help solve these problems.

The next step in the evolution of LANES is to build a generalized LACN expert system framework. To build a framework that is reusable, highly modular, and easily modifiable will require the base programming language to provide object-oriented features such as dynamic binding of functions (methods) to data (objects), inheritable object and methods, and object abstraction. A number of object-oriented languages provide all of these features.[1]

Framework Layers

Conceptually the framework is composed of three primary layers (Figure 4), an object-oriented database and graphical editor, a set of connectivity algorithms, and the expert system. This concept allows the lower layers to be fully reusable without the upper layers, but not the reverse since the expert system would need both the database and connectivity layers to function.

The object-oriented database and graphical editor are key elements of the system. The editor provides the main portion of the user interface with which users will create, modify, and add to networks. The database defines the relationship between network components and keeps track of component attributes such as network addresses and protocols.

1. In the computer science field, object-oriented programming is gaining popularity as a software engineering methodology and programming style. The reason for this is that object-oriented programming excels in software reusability, modularity, and ease of modification. Brad Cox discusses the concept of using object-oriented programming to develop reusable software integrated circuits, Software-ICs (Cox, Brad J., Object Oriented Programming: An Evolutionary Approach, Addison-Wesley, 1986). A commercially available product from Apple Computer called MacApp (Doyle, Ken, Wallace, Scott, and Rosenstein, Larry, *MacApp™: An Object-Oriented Application Framework*, Apple Report No. 4, Apple Computer, Inc., September 1986.) provides a highly reusable object oriented framework to implement the standard Macintosh interface, reducing both the time and amount of code needed to build applications.

Connectivity algorithms find paths between components. For fault isolation on the Goddard LACN the main algorithm is a simple depth first search (loops are not allowed in the network) along connections that support the protocols of the machines being tested. Another algorithm would be used to find "backroads" between equipment. An example would be to use a modem to bypass default network components to check the status of a computer. For other LACN configurations (e.g., a network that allowed loops), programmers could use the object-oriented capability of overriding methods to customize the algorithms.

The functions of the expert system layer are fault isolation, component monitor/polling, graphical and text fault notification, and explanations. Fault isolation is provided by both high-level and low-level methods and heuristics. Monitor and polling methods provide the capability to continuously monitor specified components or request an instantaneous diagnosis. Explanations and fault notification are built upon the user interface of the graphical editor.

Object Hierarchy

Each of these layers are built within the object hierarchy shown in Figure 5. The top node, TComponent, is the parent or superclass of all the other network object types. Each subsequent child node or subclass, therefore, inherits methods and fields from TComponent. This, for example, allows defaults to initially be set up in TComponent and later customized in subclasses as required. Some of the classes along with their methods and fields are discussed below. A more detailed design is currently under development.

TComponent has a field, fSubcomponents, which can be used to define TComponent objects as abstract entities such as a network or computer that can be broken down into its fSubcomponents. A polling method is then defined to poll these subcomponents. Another field, fConnectedTo, is a list of components that the TComponent object is connected with.

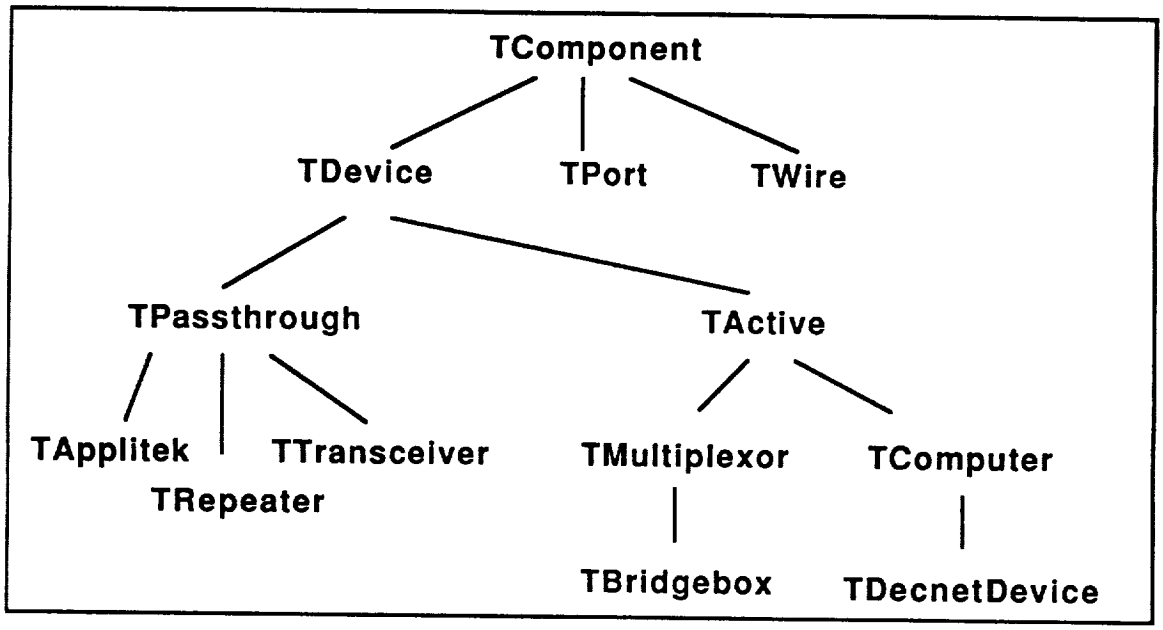

Figure 5 -- Object Hierarchy

TDevice, TPort, and TWire, are major subclasses of TComponent and define high-level information and knowledge. The TPassthroughDevice class is for devices that act as information relays and are essentially transparent to other devices on the network. During fault isolation, the up/down status for devices in this class would have to be based on the ability of TActiveDevice objects to communicate through the TPassthroughDevice object. The TActiveDevice objects can be communicated with directly to determine up/down status. The bottom nodes of the hierarchy (TApplitek, TMultiplexor, etc.) contain the most network specific information and knowledge and

is where most modifications and additions will take place.

Future

Currently a detailed design of the framework is under progress. In addition, a number of potential application areas are being explored. Once one is chosen, the hardware and software to implement the framework will be selected. Currently a variety of object-oriented programming languages (C++, Objective-C, Object Pascal, SmallTalk, LISP Flavors, ART 3.0) which exist on a number of machines are being considered.

When the LANES framework is complete, it should be applicable to a variety of LACN fault isolation problems. It could even be expanded to handle LACN design and performance analysis.

I suspect that fault isolation will not be the only area where frameworks are built. Hopefully we will see a number of frameworks being built to handle other expert system areas such as scheduling, planning, control. What will be key to their success, however, will not just be functionality, but ease of modification and expansion. Object-oriented programming will play a large part in that success.