The TROI Real-Time Operator Interface Management System for the NASREM Architecture

JOHN MICHALOSKI and BARRY WARSAW†

Robot Systems Division
National Institute of Standards and Technology
Gaithersburg, MD 20899

ABSTRACT

The TeleRobotic Operator Interface (**TROI**) management system provides a flexible, extensible, real-time interface to the NASREM [1] robot control system (**RCS**). It consists of two major portions, the Visual Operator Interface (**TROI/VOI**), and the RCS data-server (**TROI/DS**) interface modules running within an Ada concurrent programming environment. TROI provides a highly dynamic environment for interacting with the RCS. The user is able to view and modify state variables of a running control system, and to edit, save, and load graphical interface configurations while connected to a running control system. In this way, the user can interactively perform diagnostics, switch diagnostic contexts by creating and destroying interactive objects, and reconfigure data flow networks, allowing control of RCS operations without the costs of switching operator interface operating modes.

INTRODUCTION

This paper presents TROI (TeleRobotic Operator Interface), a robotic User-Interface Management System (UIMS) adapted to satisfy the operator-interface requirements for the NASA Space Station Telerobotic Servicer. The NASA/NBS Standard Reference Model for Telerobotic Control System Architecture (NASREM) [1] outlines the responsibilities of the Space Station Telerobot ***operator interface*** (OI). NASREM emphasizes hierarchical control for incorporating robot control with multiple sensors under a standard architecture.

NASREM states that the "operator interface provides a means by which the human operator, either in the space station or on the ground, can observe, supervise, and directly control the telerobot." In order to achieve this control, NASREM states that the operator and programmer interface provide the services to control, observe, define goals, indicate objects, edit both programs and data. NASREM defines a hierarchical architecture which requires that the OI share control at various levels of the hierarchy. TROI is a software toolkit that facilitates implementation of NASREM operator interfaces. Unlike many operator interface models, NASREM OI provides for operator access to all levels in the control hierarchy, not just the task-level. Plus, NASREM OI must support a variety of skill levels, from novice operation to sophisticated system support.

To satisfy the NASREM architecture requirements, TROI encompasses a broad set of complex programming concepts, including: X-Window[1] programming [10], [15]; Ada[2] programming environment [2], [5]; and the NASREM architecture [1], [6]. TROI inte-

1. The X window system is a trademark of the Massachusetts Institute of Technology.
2. ADA is a Trademark of the U.S. Department of Defense

Reprinted from PROCEEDINGS OF THE 4TH TOPICAL MEETING ON ROBOTICS AND REMOTE SYSTEMS, Feb. 1991

grates these diverse software components into a cohesive system. The TROI system features object-oriented modularity to address each of the varied programming entities. The TROI object-oriented approach maintained the conceptual separation of components while offering a convenient mechanism to manage the relationships and dependencies between the components. This consistent design methodology provides a generic OI approach adaptable to all levels in the NASREM hierarchy. However, TROI is not strictly limited to the NASREM design methodology and is readily adaptable to other real-time environments.

In real-time, TROI connects an X terminal based windowing environment via host workstation to the target Robot Control System (***RCS***). The full complement of host-based TROI tasks run under a concurrent ADA environment. RCS is a real-time, multiprocessor, shared-memory system supporting the NASREM architecture. The target and host systems communicate via a bus connector. Figure 1 shows a block diagram of the

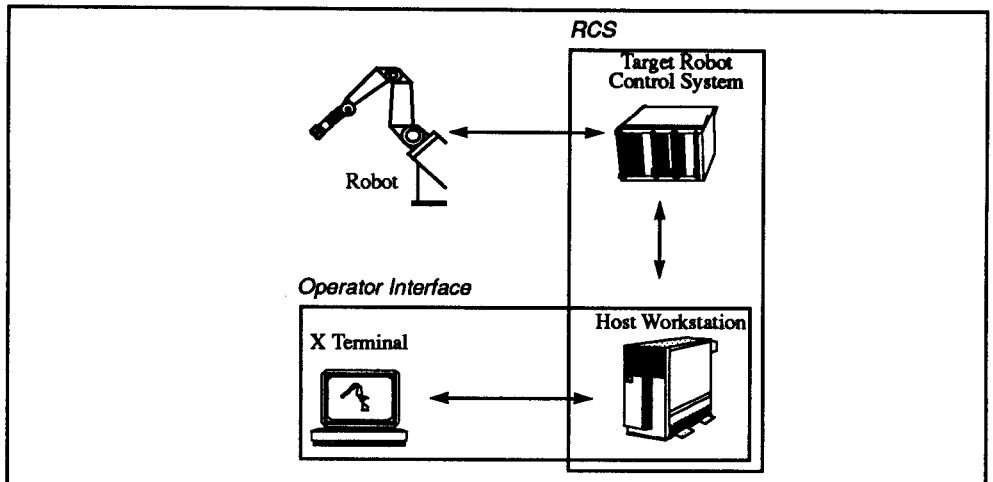

Figure 1. TROI Component Block Diagram

relationship between various components.

The fundamental TROI design principle is the separation of the application robot control data from the visual interface. A data description dictionary separates the data from the visualization. TROI consists of two processes, data routing and visual interface, that communicate via the data description dictionary. Updates to data in the dictionary were treated as transactions to be served by either the visual interface or the data server. The TROI data routing process handles real-time communication with the control system. The TROI visual interface system provides numerous functions including editing the graphical presentation, run-time transaction processing to handle data updates, and the window-based visual interface. Features and capabilities of TROI include:

- interactive edit, save, and load of visual interfaces.
- real-time (50 Hz sampling rate[1]) data acquisition and logging
- automated or manual communication connection to NASREM RCS
- scripting capability to playback recorded command and parameter sequences
- extensibility and tailorization of the data server component
- distributed visual-interface based on X Window networking
- embedded simulation testing procedure
- data server connectivity to sophisticated graphic modeling systems.

This paper focuses on TROI as a working model for real-time operator-interface technology and is organized as follows. The second section will explain the basic NASREM architecture and list desirable operator interface requirements. The third section will review relevant work in operator-interface technology as it shapes the TROI design

1. Limited by host clock precision.

methodology. The forth section will briefly cover the TROI architecture and discuss some of the larger components. The fifth section will review the interactive capabilities and visual presentation of TROI. Finally, some observations on TROI as a operator-interface testbed will be presented.

TROI DESIGN REQUIREMENTS

TROI was designed to operate within a NASREM architecture which defines a hierarchical real-time control system (RCS). The NASREM design allows for operator intervention at each level in the hierarchy. Operator intervention can be to the task decomposition, the world model or the sensory system components. Figure 2 illustrates the relationship of the operator interface at any level in the NASREM architecture. The Task Decomposition component uses a Job Assignment module to mediate autonomous versus operator control. The World Modeling component allows manipulation of global data. The Sensor System provides OI feedback of the external world.

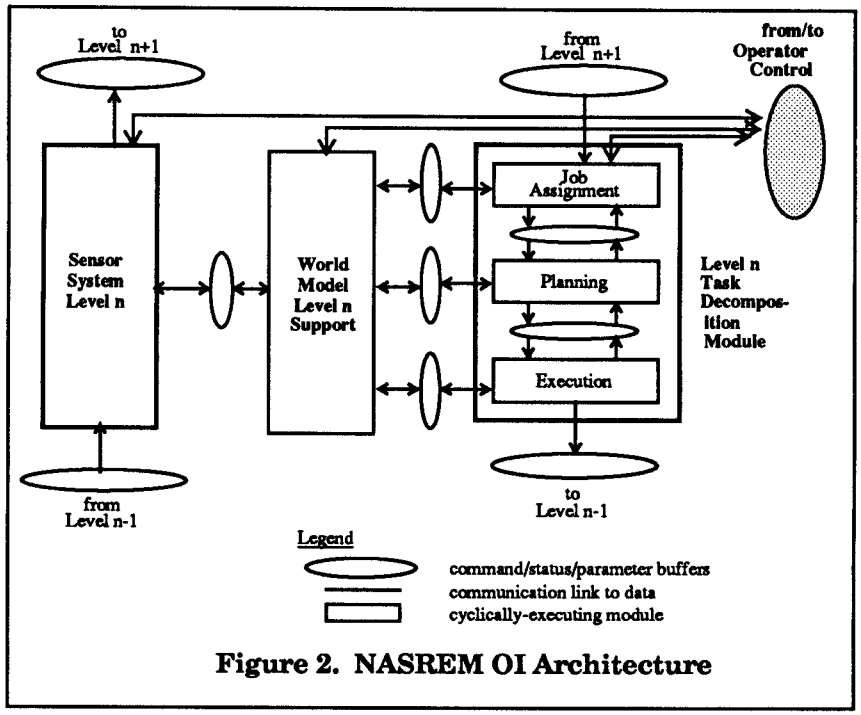

Figure 2. NASREM OI Architecture

The NASREM RCS architecture consists of a set of cyclically executing modules sharing information through predefined communication buffers. These communication buffers are designed to traffic information in a one-way data flow modeled after the reader-writer algorithm [3]; that is, one exclusive writer per communication buffer is permitted while any number of readers per communication buffer is allowed. TROI connects to the NASREM system to access and modify the command and status buffers that provide the information flow in the NASREM hierarchy. The TROI host platform is a general-purpose computer workstation that is linked to the real-time system with some high-speed hardware connection. Given the basic system architecture, the set of requirements that maximize the utility of TROI include:

Direct human manipulation allows the human operator to intrude on the operations of the control system in some structured manner. Here, the OI is providing the mechanism, rules and data paths by which human can directly manipulate, modify or override the task state space of the control system. This will incorporate the classical notions of autonomous, shared, traded, and teleoperated modes of operation.

Non-intrusive monitoring permits the operator interface to easily view the control system. Here, TROI functions analogously to a software oscilloscope, providing views of varying degrees of magnification into the control systems's operation. In addition, this requirement precludes as re-

gressive the tight coupling of display and control code that is common to embedded print statements.

Responsiveness of the system enables sampling of the robot control system in real-time for meaningful collection and display of data. This data sampling must be able to be logged either visually or to file.

Intuitive and informative visualization requires that a system be able to coherently inform the user as to changes in the environment and to flag important information for the users' attention, based on the imperative priority level of the information.

Dynamic configurability of visual interface will allow the human operator to choose from any number of situation-dependent visual configurations. In addition, dynamic reconfiguration permits a user to step through a series of visual presentations.

Modularity of design will allow changes in one part to minimally effect changes in another part. In this case, TROI consistency will be maintained if any of the architecture components, such as window system, graphic toolkit, or communication scheme, change. This requirement would make the system adaptable to changes in the technology.

Flexibility to map semantic changes in one part of the model to another. For example, a modification to the enumerated type defining the available RCS commands should directly map into a change in menu presentation.

Extensibility allows customization to specialized system and data flow needs. Data should flow automatically between the operator interface and the control system, but many times the semantic model between the user and the control system are not the same. Customization of special cases, such as the mapping of user-preferable angle degree input into control system preferred radians must be available in both data service (via callbacks) and visual interface (via filters).

Helpful systems assist the user by answering system queries and help the user to decide what pieces of information s/he are interested in and how this information can be viewed.

TROI DESIGN METHODOLOGY

Implicit with any sophisticated operator interface is the requirement for a graphical display system. Graphical images convey information in a more expressive manner than mere text, so that the selection of a graphical display technology is a major design requirement. Current graphics technology ranges from computers supporting graphic engines to personal computer windowing-systems. The disparity of the graphics technology base and the need for a standardized interface results in the need for the flexible, portable, and widely-supported Graphical User Interface (GUI) hierarchy [9].

A GUI hierarchy outlines the layers of graphical abstractions. The modularity of a GUI provides portability across a wide range of machine architectures with modifications restricted to one level in the GUI hierarchy. The levels of the GUI hierarchy from the highest to lowest level of abstraction are:

- the user
- high-level development shells
- user interface management systems (UIMS)
- graphical user-interface
    - style guides
    - toolkits
- windowing system
- operating system
- hardware

With the better understanding of the relationship among entities, operator interface technology has evolved into a state that it is now crystallizing into a science [6]. For this reason, it was difficult to decide whether it was better to build a system or use a commercially available one to meet our graphical needs. We evaluated a number of available

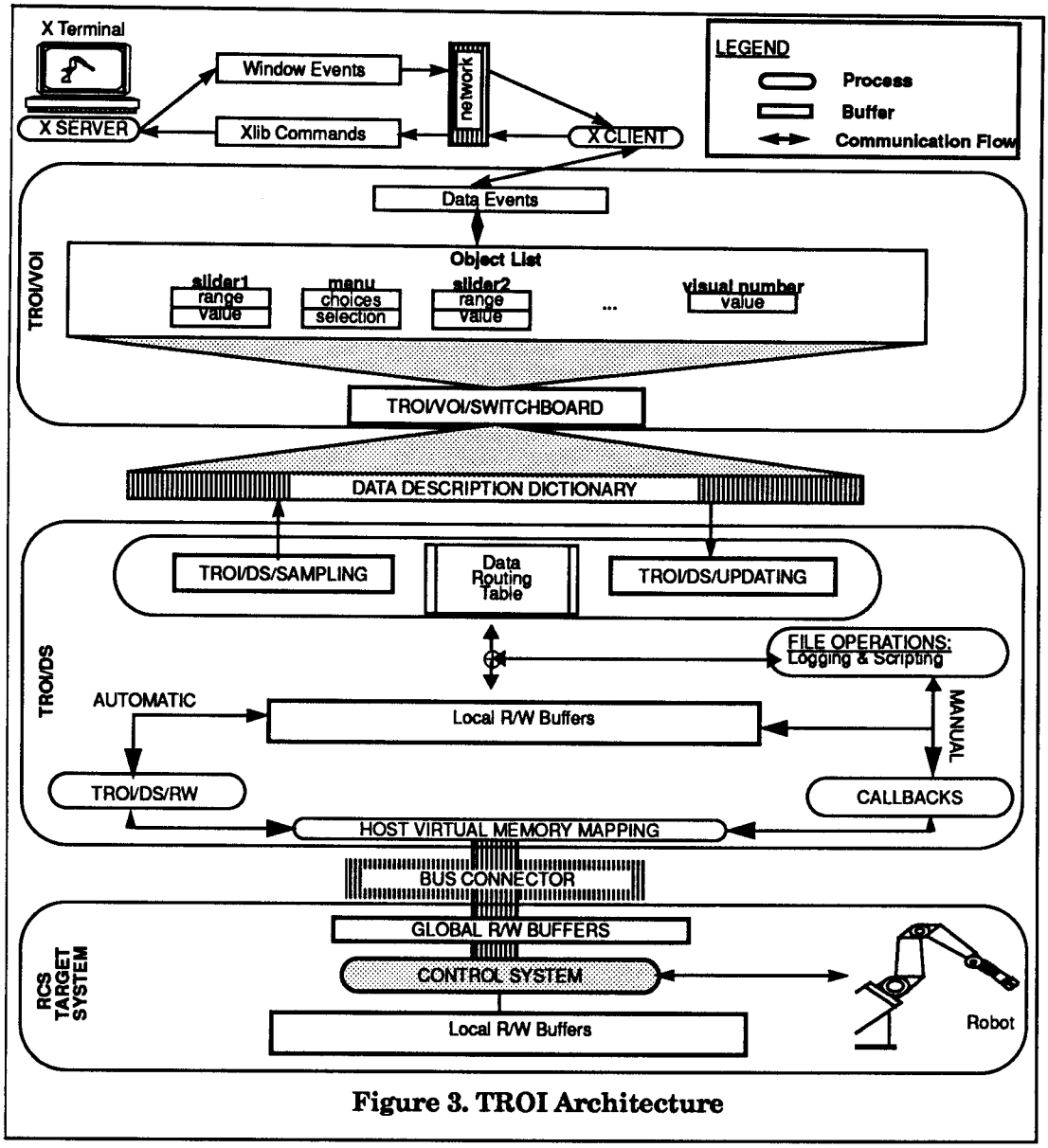

Figure 3. TROI Architecture

off-the-shelf systems. Nearly all lacked a coherent real-time data service strategy between the control system and the operator interface.

Drawbacks to commercial systems were varied. One system style allows interactive development and editing of the user-interface (also known as rapid prototyping) and then produces source-level code for integration into the application code. This suffers from the lack of flexibility to map changes made in the application user-interface to the interactive user-interface development model. Other systems offered the rapid prototyping feature with hooks for connecting the data flow but were limited by the lack of real-time data communication channel, using either files or UNIX[1] "sockets" as the protocol.

SERPENT, under development at the Software Engineering Institute, Pittsburgh, Pa., addressed many NASREM GUI requirements. SERPENT completely decouples the application program from the user-interface so that communication between the two is done through an intermediate language. SERPENT provides both an interactive visual editing and a transaction-based handler of run-time data flow. Although robust, SERPENT was incomplete at the time of the TROI design process.

1. AT&T Trademark

Without a viable off-the-shelf option, we designed the TROI GUI system to accommodate the computer systems in our lab, yet remain sufficiently portable to other platforms. The host computer platform dictates several GUI elements. TROI uses the engineering workstation as the computer platform. Most engineering workstations provide networking capabilities which implies the need for a networked windowing system. The X Window System[1] developed at MIT is widely-supported, public domain software that provides graphical networking [10]. Although X does not feature high-resolution graphic support, X has such broad industry support that it has become a de facto standard. Although we could have chosen a specialized graphics platform supplying better visual rendering for interface operation, we felt such systems lacked the standardization that addressed our needs for portability.

X Window System

The X Window System is based on a client/server model. The X server supplies the primitive graphics routines while clients send packets of instructions to the X server for graphical display. The client/server model of X provides distributed graphical communication so that client graphic programs can run across machine boundaries.

The basic data object in X is the widget which encapsulates graphical information with an object-oriented approach. Sets of widgets are defined as toolkits. To the application programmer, a toolkit is a library of graphical data abstractions, supplying objects with methods. A sample of these objects include menus, slide bars, panels, buttons and alert fields. A style guide is responsible for defining the toolkit visual representation, for example, the menu widget displaying a pop-up style as opposed to a pull-down style.

Within the X world, numerous toolkits are available to augment the functionality of X. TROI uses the XVIEW[2] toolkit which presents an OPEN LOOK[3] style window manager [7], however, because of the insularity of GUI hierarchy, transition to another window-manager style is possible.

User-Interface Management System (UIMS)

Although helpful, managing the visual objects with an X toolkit is time-consuming and specialized. Minor changes in visual presentation do not map into minor programming changes. For this reason, the separation of application data from graphic visualization is imperative in a robust and flexible operator interface. Within the GUI hierarchy, it is the UIMS level that is responsible for handling the requirement for the separation of application and user functionality. After a decade of existence, UIMS technology has finally gained acceptance as the prescribed methodology [8].

Managing visual objects in a UIMS is similar to handling data in a DBMS. Management systems separate the physical representation from the logical representation. The UIMS abstraction layer provides general system organization and at the same time offers flexible construction techniques. A robust UIMS design decouples the application program from the user-interface. A UIMS has two major responsibilities, constructing the visualization and handling the run-time data and event flow. An interactive UIMS editor provides fine-tuning adjustments necessary in defining the user-interface. At run-time, a UIMS transaction process handles communication between the application and the user.

TROI ARCHITECTURE

The UIMS design issue to separate data and visual representation molded the TROI architecture. TROI achieved separation through the use of a data routing server connecting the control application to the user-interface. Another major design issue was the

1. MIT Consortium Trademark
2. SUN Computer, Inc. Trademark
3. AT&T Trademark

desire to streamline performance in order to maximize system responsiveness. For this reason, the control application and the user-interface communicate through an intermediate data description dictionary, as opposed to an intermediate language. The dictionary was conceptually simpler, plus less of a performance burden. Further performance benefits were derived from unifying the dictionary and TROI components under a single, concurrent, ADA program space as opposed to a distributed machine space. Single execution space provided a conceptually straightforward communication, synchronization, and execution model.

TROI features a multi-layered software architecture. Figure 3 graphically depicts major components of the TROI architecture and outlines the functional layering from the user to the control system. At the user level, TROI Visualization Operator interface (***TROI/VOI***) supports visualization through a workstation window-system. The TROI/ VOI component is responsible for providing visual services. The Data Description Dictionary (***DDD***) contains data format descriptors. The TROI Data Server (***TROI/ DS***) connects the user-interface to RCS. Reader-writer buffers provides communication between the data server and the control system.The NASREM reader-writer strategy uses global data buffers for communication and local buffer images for manipulation. The local buffer images are written/read from the global data buffer during communication.

TROI Visual Operator Interface

TROI/VOI provides front end access to the list of DDD variables extracted from RCS. It is through manipulation and display of these variables that the user is able to change the internal state of the RCS, and gather information about the state of the RCS. To achieve this, TROI/VOI handles the visual interpretation of data, the interactive modification to the display layout, connectivity of data to visual representations, and run-time input/output transaction processing.

The basic element in TROI/VOI is the visual object. The visual operator interface is constructed by creating, modifying and destroying visual objects. TROI/VOI depended on the XVIEW toolkit to provide the basic set of objects and methods. Such objects include: menus, sliders, panels, text input/output fields and buttons. Robot control needs additional, more specialized, graphical objects that are not identifiable to windowing toolkits. The set of more specialized robotic include such graphical objects as: 2-D plots, bar graphs, histograms, trace graphs and table entry mechanisms. TROI/VOI constructed these graphical objects using X graphical primitives.

TROI/VOI data flow is based on pairing ***producers*** and ***consumers*** of data. The terms consumer and producer are used to describe either DDD entries or visual objects whose data flow direction have been determined. For example, a slider visual object can be used as either an output descriptor or an input mechanism. An interface design decision establishes the slider connection as either a producer of data to the control system, or a consumer of data from the control system. A single producer may be connected to any number of consumers, and each consumer will see an exact replica of the information at each clock tick. A single consumer may only consume information from one producer. The structure which maintains the data flow connections in the VOI is called the ***switchboard***. The switchboard connections are established interactively with the TROI/VOI editing capability. At run-time, the switchboard can be dynamically reconfigured to achieve new visual presentations and data modelling.

The VOI is both data driven and event driven. As data changes, the VOI recognizes this and propagates these changes along the data flow path. As the user interacts with visual objects in the interface configuration, these window system events are recognized and acted upon. TROI?VOI merges an X window system event loop with a TROI/DS data event loop.

Data Definition Dictionary

Building a TROI system depends on the user defining the set of important RCS variables that must be available for the user to manipulate. The DDD contains the list of

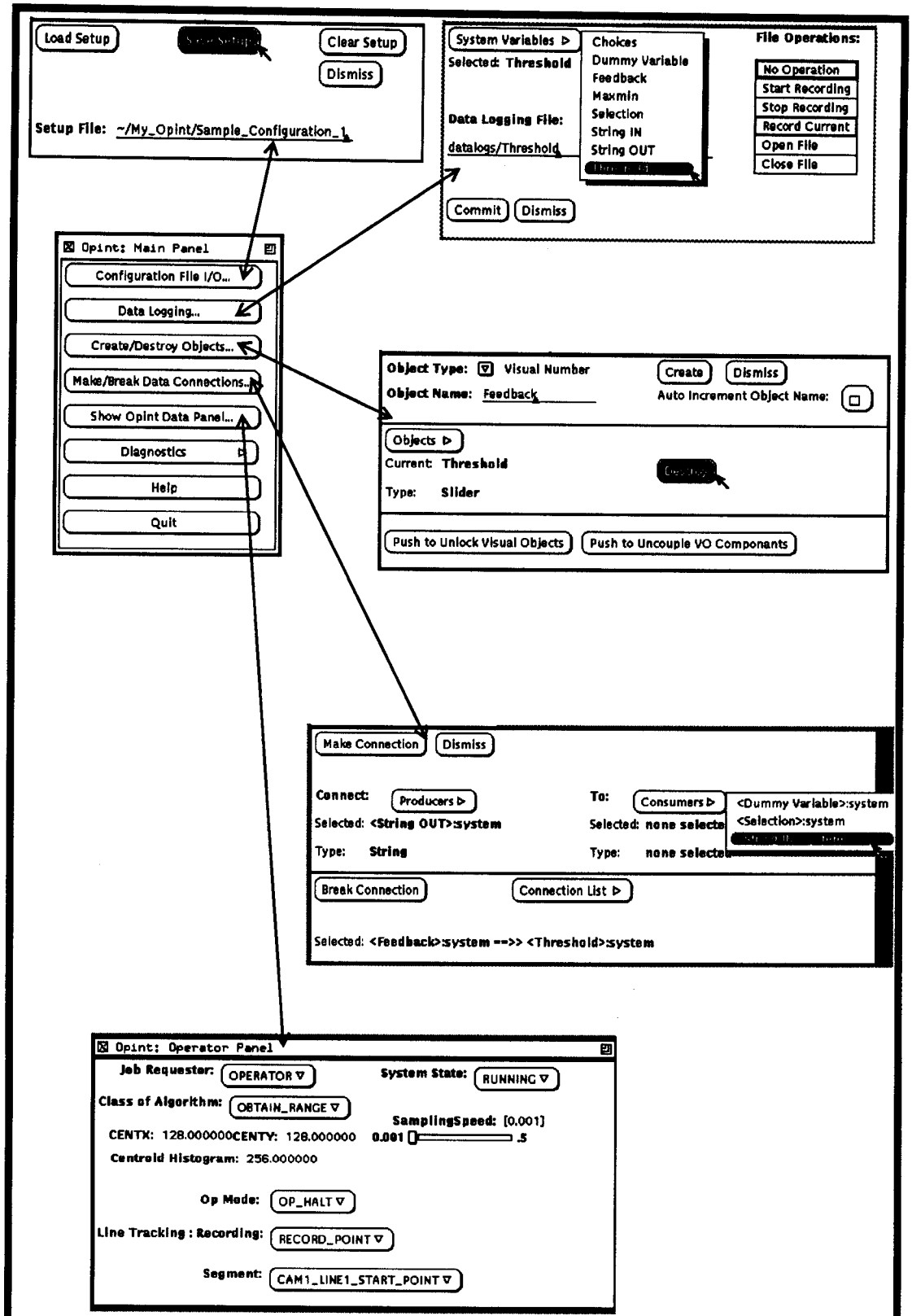

Figure 4. Interactive TROI Configuration Development Session

data that can be routed through the system built from a set of predefined data formats. These data formats rely heavily on the ADA default parameter feature to simplify the interface. With defaults, only partial specifications are needed upon invocation. The partial specification allows a more natural style of programming geared towards specific needs, where default parameters are inherited from the basic data format descriptors.

The current data descriptors mimic the basic data types one would find in the control system. The set of generic data descriptors include: n-dimensional numeric vector, enumerated list for choices and selection, boolean, string, alert/error/informative text, and reader/writer command increment based on buffer update. Some special descriptors were developed to map spatially diverse data into a contiguous representation. For example, a 2-D plot may require pairing variables from one record with a possibly different record to achieve the x,y representation.

At run-time, DDD contains the latest transaction updates to be processed. TROI uses DDD as the communication buffer between TROI/VOI and the TROI/DS and with a set of signal routines coordinates data flow between control system and user-interface.

TROI Data Routing Service

The TROI/DS handles the routing of data to and from RCS through the NASREM reader-writer communication mechanism. TROI/DS has two purposes: move data out of RCS (sample); and move data into RCS (update). TROI/DS/SAMPLING is responsible for cyclically sampling RCS status buffers, and upon arrival of new user-interface directives, write RCS command buffers.TROI/DS/UPDATING is responsible for updating RCS reader-writer buffers upon either external input from the user or under scripted sequencing. TROI/DS uses the data routing table to resolve physical addressing specifics of the control system reader-writer communication interface. TROI/DS supports file operation as a part of the data routing function. File operation allows data logging and retrieval independently of the visual interface.

TROI supports both ***automatic*** and ***manual*** target system communication modes. To achieve an automatic TROI system, one simply defines the DDD, connects data to visual objects, and then starts TROI. Each data update is automatically transmitted between subsystems. Many systems need only a one-to-one variable update correspondence. There are times when an entire series of variables must be set before transmitting the set of variables through the communication buffer. For example, a command buffer to a robot may require a command, a set of joint or Cartesian values, and a traversal time. Sending each of these variables upon update is questionable. In manual mode, the user sets up all command values, and then upon signal, the command is actually transmitted via a manual write of the reader-writer buffer to the target system.

Callbacks are a TROI/DS feature provided to allow extensions and are considered part of the manual operating mode. These extensions allow a broad range of customization and are powerful in addressing application-specific requirements. At DDD definition, the user supplies a callback procedures. Callbacks (to the defined procedure) occur every time the DDD entry is updated. Callbacks are useful for implementing such features as:

- filters. For example, converting a user-defined angle into radians before sending to RCS.
- broadcast. When one variable is set, subsequently setting other companion variables. For example, pushing one button might set three board flags.
- synchronization. When a condition occurs, initiate some action. For example, when a status variable reads done, allow a new command to be sent.
- playback/scripting. In combination with synchronization, a variable can be used to initiate playback of a script from a file by sending the recorded sequence of commands to the target system. After initiating, the synchronization feature could send subsequent commands.

TROI RUN-TIME MODEL

In TROI, the control application programmer is responsible for selecting the control system variables for data routing. All data routed through the system is placed in the dictionary. Description entries in the dictionary define the format of the data, physical location in the control system, and any file connections for data logging and scripting. Using the TROI user-interface, the application programmer (or graphics design expert) independently decides the visual format of dictionary data entries. An interactive editing session defines the visual interface beforehand. At run-time, individual data is entered or displayed according the visual format definition. TROI user-interface also supports changes to the visual representation at run-time. For example, while attempting to troubleshoot a problem an operator may decide that a histogram plot conveys more information than an instantaneous bargraph.

Different needs and capabilities resulted in the implementation of TROI with a number of programming languages. The control system uses an ADA development environment. TROI uses a mixture of ADA, C and Assembly. The TROI/X module was written in C because the X environment is tailored for C. The TROI/DS was written in ADA since RCS and TROI/DS share data types. DDD is a list structure that has an isomorphic ADA and C definition.

At run-time, TROI consists of three concurrent Ada tasks including TROI/DS/SAPMLING, TROI/VOI, and TROI/DS/UPDATING. Each of the tasks runs asynchronously and communicate through signals. An optional ***simulator task*** is provided for debugging under a completely host environment. Simulated mode transparently substitutes reader-writer communication to host-resident dummy buffers instead of addressing the target system. The simulator task reads and writes from these dummy communication buffers that would normally be sent to the target RCS system. The simulator is useful in testing a TROI system and saves much time and effort when the final target interfacing is performed.

In order to get a feel for the TROI operator interface, the following sections will cover some management operations available.

TROI/VOI Control Panel

The Control Panel is the window containing 7 buttons including *QUIT* within Figure 4. Using the Control Panel one orchestrates a TROI session. The *Show Opint Data Panel* button pops up the main run-time *Opint: Operator Panel* operator-interface window. The crux of a TROI operator interface is constructed within this window.

Interactively constructing a TROI operator interface requires several phases. One needs to layout the visual objects (*Create/Destroy Objects...* button), connect visual objects to DDD entries (*Make/Break Data Connections...* button), and then save the configuration (*Configuration File I/O...* button). This construction process can be performed all at once, or incrementally. Once the TROI operator interface is configured and saved; from then on, data flow connection to the RCS system is available.

Other panel buttons provide self-explanatory service. For example, the *Data Logging..* button pops up the window to configure data logging.

TROI Operator Panel

The Operator Panel (window labeled *Opint: Operator Panel* within Figure 4) contains the set of visual objects that will communicate with the control system. After configuration, most user interactions will occur within this frame. Once configured, automatic window configuration and display can happen at session startup circumventing the need for *Control Panel* button pushing.

TROI Configuration Management

Once a TROI configuration has been set up, it can be saved to a file and reloaded during a future session, or at a later time in the current session. This way, you can have

multiple data flows and interfaces corresponding to different sub-tasks. Configurations can be merged by loading one setup file after another, or you can clear the current configuration before loading the new one. All visual information pertaining to the current configuration is saved in the setup file.

Visual Object Definition

In this section we will discuss TROI *Object Frame* (shown in Figure 4) approach to creating, positioning and destroying objects. This frame is divided into three subpanels: the top panel is used to name and create new objects, the center panel is used to destroy already existing objects, and the bottom panel is used to control positioning of visual objects in the *Opint: Operator Panel.*

To create a new object, you must specify the object's name and the type of object to create. The object type is selected through the pulldown menu entitled *Object Type* at the top of the Create subpanel. A unique object name must be entered in the text field below the *Object Type* menu. When the *Create* button is pressed, an object of the selected type and name is created. The auto-increment feature allows automatic naming for quick system prototyping.

To visually configure the screen layout, one repositions visual objects. Initially, visual objects are locked in their default positions when created and must be unlocked (with the *Push to Unlock Visual Objects* button) before they can be repositioned. Also, most visual objects are aggregates of sub-components and these sub-components must be uncoupled (with the *Push to Uncouple VO Components* button) before they can be repositioned.

Data Flow Connections

At any time during TROI configuration building or run-time, the Connections frame (Figure 4) is used to make or break data flow connections from producer variables to consumer variables. By making and breaking data flow connections, you establish data paths which propagate information through the system.

The Connections frame has two sub-panels: the top panel is used to make data flow connections and the bottom panel is used to break data flow connections. Before a connection can be made, both ends of the data connection must be selected. Pop up menus are attached to the *Producers* menu button (containing the producers list) and the *Consumers* menu button (containing the consumers list), to select terminus of the connection. In Figure 4, the user is in the middle of the selection process. A producer variable has already been selected and the user is about to select the consumer variable for the other end of the connection.

Variables of different type cannot be connected together and a variable of an undetermined direction cannot be connected to itself. Once both a consumer and a producer are selected, pushing the *Make Connection* button will attempt to attach the two variables. If the connection is unsuccessful, an error message will be displayed to indicate the problem. If the connection was successful, the consumer will be removed from candidate list.

Data Logging

System variables supplied by RCS can be ***data logged*** to a file for examination. TROI/DS handles the actual data logging operations for which the VOI provides a convenient front-end. These functions are accessed through the *Data Logging* frame as shown in Figure 4. An alphabetical selection list of the DDD entries is presented to the user in the *System Variables* pull right menu. Figure 4 shows the user in the process of selecting the *"Threshold"* system variable for data logging. The name of the data logging file to which values will be stored is entered in the *Data Logging File* text input field or can be declared as a part of the DDD entry definition. The file operation to perform on the selected system variable is chosen from the *File Operations* menu. Only one operation can be selected at a time. The available operations are:

Start Recording: Informs TROI/DS to begin recording a data series of the selected variable. Data logging will continue on this variable until a *stop recording* operation is performed.

Stop Recording: Informs TROI/DS to stop a data series recording.

Record Current: Inform TROI/DS to record only the current value of the selected variable.

Open File: Informs TROI/DS to open the data logging file as specified in the *Data Logging File* text input field.

Close File: Closes the selected data logging file previously opened with an *open file* operation.

Data logging operations can be on-going for more than one variable at a time. The *System State* menu button in the *Opint: Operator Panel* can be used to synchronize the start of multiple data entry logging. If several variables are to be logged simultaneously, the user selects the variables, opens the files, starts the recording, but actual recording of data will not start until the *System State* menu button changes from *IDLE* to *RUNNING*. Further, the user is allowed to specify whether these data loggings are time stamped.

SUMMARY

Building the operator interface for any application is time-consuming. TROI was designed to simplify the construction and management of real-time NASREM robotic operator interfaces. TROI adds a layer of transparency between the visual interface and the control system by providing a management system full of helpful tools to interface to different system components.

Our goal was to achieve flexibility in response to a variety of requirements. TROI provides a wide-range of services suited for different levels of control. As we have applied TROI to different scenarios, we adapted TROI to the new needs. For better visual rendering, we extended TROI to communicate with a graphics-engine workstation. Many current adaptations are enabled as part of the callback routine from the TROI data service. As it is apparent that these are not really special case, but rather, general needs, features enabled through callback routines will be incorporated into the general TROI management system.

REFERENCES

1. Albus J.S., McCain H.G., Lumia R. "NASA/NBS Standard Reference Model Telerobot Control System Architecture (NASREM)", NBS Technical Note 1235, National Bureau of Standards, Gaithersburg, MD (June 1987).
2. Booche, Grady. Software Engineering with Ada, (Benjamin/Cummings Publishing, Menlo Park, CA 1987).
3. Brinch Hansen, Per. Operating System Principles (Prentice-Hall, Englewood Cliffs, NJ 1973).
4. DoD Reference Manual for the Ada Programming Language, ANSI/MIL-STD-1815A-1983.
5. Fiala, J.C. "Note on NASREM Implementation", NIST Internal Report 89-4215, National Institute of Standards and Technology, Gaithersburg, MD (December 1989).
6. Hartson, H.R., Hix, D. "Human-Computer Interface Development: Concepts and Systems for Its Management", ACM Computing Surveys, 21(1), 5-92 (March 1989).
7. Heller, Dan, XView Programming Manual, An OPEN LOOK Toolkit for X11 (O'Reilly & Associates, Sebastopol CA 1989).
8. Hix, D. "Generations User-Interface Management Systems", IEEE Software, 77-87 (September 1990).
9. Mandelkern, D. "A GUIDE to High-Level User Interface Development Tools", SUN Expert Magazine, 1 (3), 71-76 (Jan. 1990)
10. .Nye, Adrian, Xlib Programming Manual for Version 11 (O'Reilly & Associates, Sebastopol CA May 1989).