Module Interconnection Frameworks for a Real-Time Spreadsheet

Final Report

Program: SBIR N92-112

Scientific Officer: Ralph Wachter, Office of Naval Research

Principal Investigator: Richard Clarke, RTware, Inc.

Date: October 19, 1993

2.0. Abstract

Phase I research into a modular real-time spreadsheet has yielded a significant result: A spreadsheet can be used to completely simulate, completely prototype and completely implement distributed control systems. A usable implementation of such a system will be done in Phase II. Commercialization will begin in the middle of Phase II, with the release of a set of distributed processing options for RTware's ControlCalc product. Demonstration applications in Naval applications will be done in cooperation with RTware's existing Naval customers.

A real-time spreadsheet will be used to build nodes in a software module interconnection framework (MIF), allowing interactive, on-line construction of distributed control applications. Due to major advances in spreadsheet software technology already implemented in ControlCalc, the commercially-available spreadsheet to be used, the system will allow prototyping, simulation, top down design and implementation down to the final, lowest-level runtime machine code. The MIF is Polylith, which provides a remote function calling and message-passing abstraction that can be used by any language, including a spreadsheet. A graphical diagram editor will provide visualization of and navigation through the module framework.

The major spreadsheet advances are: on-the-fly expression compilation, multi-threading evaluation, and multi-tasking text or graphical user interface sessions. Important supporting capabilities include: in-line I/O functions for direct access to analog or digital hardware and file or network protocols, strong typing, direct mapping to the output of graphical user interface editors, and extensibility through calling user-supplied functions in other languages directly from compiled spreadsheet expressions.

Statement A per telecon Ralph Wachter
ONR/Code 1133
Arlington, VA 22217-5000
NWW 12/15/93

2.1. Introduction

A commercially available real-time spreadsheet will be integrated into a module interconnection framework (MIF), creating a distributed control system (DCS) with major improvements in programming productivity and system performance over existing software technology. A consistent, high performance DCS running on a wide range of heterogeneous computer platforms will have application in Naval and commercial software programs ranging from embedded control to data center operations. Existing Naval applications using RTware software with Navelex and the Naval Research Laboratory will be used as demonstrations of the resulting product.

The spreadsheet package will be ControlCalc from RTware. Uniquely among spreadsheets, ControlCalc evaluates compiled code in a multi-tasking, shared memory system organized in a three-dimensional spreadsheet format. ControlCalc is already in use in demanding commercial and DoD applications such as turbomachinery control, mainframe monitoring, and airfield control tower control.

The MIF package will be Polylith from the University of Maryland. Developed under DoD contracts, Polylith is a software bus that supports network message passing and high level interface definitions between software modules running on heterogeneous computer systems. Language independence and a module interface language and compiler allow spreadsheet modules to communicate with modules implemented in languages such as C which have a Polylith interface. Easily portable, Polylith is in use already in distributed applications.

The Phase II work plan involves three major stages: (1) DCS-I, (2) DCS-II and (3) a graphical programming interface. DCS-I will build on initial design and prototyping work from Phase I to let ControlCalc users interactively declare module interfaces and invoke remote services over Polylith directly from aliased spreadsheet functions. DCS-II will use both Polylith and shared memory technology to allow the distribution of threads of control within the spreadsheet matrix over tightly or loosely coupled processors. The graphical programming interface will let users construct applications at the module level, with hierarchical module creation and run-time application visualization.

DCS-I will support interactive definition of module interfaces and the binding of interface parameters to spreadsheet data cells, thereby enabling functioning prototypes to be rapidly constructed. Each module would be implemented as a separate spreadsheet, capable of running on any supported platform. The spreadsheet interface allows designers to view live data, interactively change data to test the interfaces and construct simple spreadsheet functions that simulate the intended operation of the module. As the spreadsheet functions are as efficiently evaluated as functions in any other compiled language, such simulations can be interactively enhanced until they completely implement a module's specification. In fact, the spreadsheet can be considered a specification language, since functions, comments, task scheduling declarations, I/O declarations, report formats, etc. can be entered into the spreadsheet in readable form, yet directly executed by the spreadsheet at compiled speeds. It should be noted that spreadsheets are very heavily used in commercial applications for their modeling and what-if capabilities. DCS-I will result in commercial application immediately, through the release of an enhanced ControlCalc product with a Polylith option. Demonstration projects with DCS-I with Navelex will show immediate commercial applicability.

DCS-II will provide much higher performance than existing DCS systems that use a data-base paradigm with message passing. The ControlCalc compiler will use module interconnection declarations to resolve inter-module references, keeping the spreadsheet data matrix in shared memory when possible. Real-time synchronization of distributed concurrent modules will be accomplished by allowing module interfaces to declare monitor, signal and wait operations. Implementation will use ControlCalc's existing semaphore-based synchronization primitives. Commercialization of DCS-II will focus on the DSP programming market, using the NRL demonstration project as a motivating example.

The graphical programming stage will provide block functions with graphical front-panel displays that can be created by users with spreadsheet arrays mapped directly into graphical object attributes. Existing ControlCalc declarative techniques for mapping graphical attributes to spreadsheet tabular data will be extended to create an object-oriented graphical environment. Today's controls and automation market has a high demand for graphics, both in programming and end-user interfacing. ControlCalc's existing techniques for driving graphical displays and graphical editors have proven very popular. Enhancements in this area will be immediately commercialized as soon as consistent sets of features are implemented.

In general, commercialization of the resulting product will be done through RTware's control systems business and through major corporate partners and OEM's such as Motorola Corporation, Encore Computer Corporation and Gespac S.A.. Successful ControlCalc applications in DoD and private industry have validated the real-time spreadsheet paradigm. Integration of ControlCalc technology in MIF will allow large-scale applications to make incremental and reliable transitions to the spreadsheet paradigm where appropriate, while preserving existing software investments. Important Navy and commercial applications with critical distributed real-time requirements will be able to achieve the productivity gains already demonstrated by ControlCalc in high-speed multi-tasking applications.

Letters of support for the Phase II project from existing ControlCalc partners and customers, including Naval applications, are attached in section 8.4.

3. Identification and Significance of the Problem or Opportunity

3.1. Summary

The software productivity problem presents the opportunity for large cost savings if a software development environment can deliver three critical capabilities:

1) Modular Design with Simulation and Prototyping
2) Effective Interconnectivity
3) Sophisticated Design and User Interfaces

A modular interconnection framework must be simple and flexible enough to allow existing software to be easily packaged into a modular format. New languages providing fast simulation, prototyping and implementation must be able to interoperate with older languages through a module interface. Module interfaces must be quickly constructed, tested and profiled so the sufficiency and efficiency of an application's modular composition can be determined early in the design cycle. Hierarchical module libraries must be created which encapsulate generally useful functions for use in multiple applications.

The Phase II efforts proposed here will combine the ONR-sponsored Polylith MIF system with RTware's commercial ControlCalc spreadsheet system to provide exactly those capabilities in a distributed control system. The productivity savings opportunity will be proven by applying the resulting system to actual control applications in use by existing ControlCalc customers, including two U.S. Navy projects, one through the NRL and one through Navalex.

This is a critical juncture in ONR's MIF research program, a point at which Naval application developers must be motivated to use MIF through concrete demonstrations of its advantages. A demonstration that MIF allows as radical a new programming concept as ControlCalc to be integrated into existing Naval applications will provide that motivation. Commercial success of the technology is guaranteed, as the ControlCalc system is already proven in commercial applications to have significant competitive benefits in performance and ease of use. Furthermore, major new commercial opportunities will be realized, since the MIF capabilities required by the Navy are also important in the commercial distributed control systems market. The work proposed for Phase II directly addresses the technical issues which RTware's market experience indicates are important in the further development of the ControlCalc product.

3.2. Detailed Description of Problems and Opportunities

This section provides detailed discussions of the problems and opportunities, including descriptions of the two software systems involved, Polylith and ControlCalc.

3.2.1. The Problem Statement

With the increasing level software functional requirements and the large cost-performance advantages of multi-processing hardware, the level of complexity in software systems is increasing exponentially. Particularly within DoD control systems, requirements for redundancy, recoverability and reliability increase demands on software systems which must also maintain the highest level of algorithmic sophistication possible in their mission-critical, competitive environments. The result is a rapid expansion in software development costs, and increasing difficulty in proving system reliability. Software engineering techniques that provide measurable increases in productivity will provide very large benefits in reduced cost, improved functionality and faster deployment.

3.2.2. Modular Design

The primary productivity technique for solving the software complexity problem is modular design. Modular design allows components of a system to be specified in terms of the interfaces they present between each other and the internal transformations they make on data passing through those interfaces. Once defined, modules can be implemented and thoroughly tested without requiring completion of the entire system. However, it is often necessary to prove out low-level functionality before the feasibility of a high-level design can be assured. This means that there is a need for rapid implementation of low-level modules and flexibility in the design of their interfaces.

The opportunity addressed by this proposal is to provide high-level modular design tools for distributed systems, while also providing a simple and productive programming system for internal module design.

3.2.3. Module Interconnection Framework: The Polylith MIF

A Module Interconnection Framework (MIF) is a system which allows software modules to connect to each other using a formal interface structure. Polylith is a MIF system developed at the University of Maryland under ONR and DARPA contracts. Polylith's interface structure is independent of the language and type of computer used to implement the module. Polylith presents a consistent view of data types, along with automatic data conversion as required. The interconnection mechanism has general-purpose message-passing and client-server options, and is presented as an abstract software "bus". A module specification language (MIL) is used by developers to declare module interfaces without regard to the module implementation language. A MIL compiler builds the bus from a set of module declarations.

Due to the language independence of the Polylith system, incremental use of the spreadsheet module construction system will reduce the risk of adopting that new design methodology. The simplicity of porting Polylith and of wrapping existing code in Polylithic modules means that there is a low cost for adding Polylith capability to existing software systems.

3.2.4. Advances in Programming Productivity: The ControlCalc Spreadsheet Paradigm

Commercial control applications currently running with ControlCalc demonstrate that major productivity gains can be realized through the use of spreadsheet techniques. This is particularly true in continuous processing, real-time types of applications. A spreadsheet can be considered a collection of rules operating on a data space, or a simple functional language. Continuous processing real-time applications typically acquire data, apply a set of rules and output the results. Inputs and outputs can both be real-world sensors and actuators, in which case it is a feedback control system. Inputs can be real-world data and outputs be data storage and human visualization, in which case it is a data acquisition system. A number of significant application cases demonstrate that a spreadsheet is a natural tool for specifying continuous processing rules.

3.2.4.1. ControlCalc's Design: A Competitive Advantage

ControlCalc is a spreadsheet designed for rapid implementation of the all levels of real-time applications by the control or software engineer. With ControlCalc, the application development process is done within the spreadsheet, without sacrificing performance or functionality. ControlCalc achieves that result with four major innovations:

1) On-line compiling.
2) Continuous multi-tasking evaluation in a shared memory model.
3) Direct real-world I/O functions.

4) Expanded data types and functions

Real-Time performance comparable to 'C' programs is achieved by embedding an on-line compiler that converts spreadsheet functions to highly efficient machine code. On-line, embedded compilation directly into executable memory is fast enough to maintain the interactive feel of a spreadsheet.

ControlCalc is a three-dimensional spreadsheet and provides multi-tasking evaluation of each page as a separate task or thread. The three-dimensional spreadsheet matrix is kept in shared memory, and acts as an on-line matrix data base directly referenced by the compiled spreadsheet expressions. The spreadsheet application designer declares the trigger mode (cyclic, event-driver, interrupt service routine) and parameters of each page allowing the ControlCalc run-time executive to start, stop and manage running the application. The user-interface part of the spreadsheet is on-line at all times, allowing the user to move around and inspect any expression's results while the system is running. Manual run control, on-line source statements and very minimal edit and compile turnaround times make the spreadsheet an effective debugging and prototyping system.

Direct real-world I/O is accessed by I/O functions that are part of the standard spreadsheet function set. The user declares logical I/O port numbers and selects from a list of supported I/O hardware, including analog, digital, counter/timers, and network options. Unique buffer cells (fifo or lifo) support large data sets without the per-point spreadsheet cell overhead, with DMA drivers for acquisition and file operations.

ControlCalc expands the traditional number/text data types of a spreadsheet to include integers, real numbers, booleans, variable text strings, constant text strings and array buffers. Each type comes with a full set of functions and operators that make ControlCalc a full-featured programming language.

3.2.4.2. ControlCalc Productivity Demonstrated

In ControlCalc applications, a controls consultant indicated that large state-table driven algorithms for chemical process control could be expressed reliably in a spreadsheet in a matter of hours, while traditional control languages required weeks of implementation and testing. In another application, a process control engineer doing multi-loop (60 task) control of high-speed motors reports major productivity improvements, particularly in program maintenance, when compared with traditional control systems from a leading international controls company. His project involved two years of side-by-side pilot projects in actual production on their natural gas pumping stations. The project resulted in the selection of the ControlCalc spreadsheet for future turbine control systems over one of the most advanced programmable controllers on the market today. Finally, a system monitoring and adaptive tuning application for large-scale multi-processing UNIX mainframe-class systems was developed with ControlCalc and deployed in less than one person-year. With over 200 display screens, 1000 I/O points per node and 30 different major tasks. this system is judged superior to competitive systems that required dozen's of person-years of development.

3.2.5. Distributed Control Systems: The DCS problem

Distributed Control Systems (DCS) have been in use in private industry for over twenty years. A DCS system is a loosely coupled processor network usually involving supervisory computers and dedicated, proprietary programmable controllers. DCS systems have three major problems: interoperability, performance and a limited domain.

Lack or interoperability in DCS systems is a problem because DCS systems are proprietary. Each manufacturer has their own controller, control programming language, network hardware and network protocol software. Efforts to standardize within the industrial market have failed, particularly the MAP

(Manufacturing Automation Protocol) promoted by General Motors. While standardization efforts continue with the industrial network concept called FieldBus, two major groups have already split off, ISP headed by Seimens and WorldFIP with Honeywell and Allen-Bradley. Furthermore, neither of these specifications addresses the domain of modular programming, focusing instead on communication with intelligent I/O devices.

DCS performance has proved to be a problem due to the database transaction approach that underlies their design. Schelberg, a leading control software designer, identified file and database access as limiting factors in DCS performance and proposed virtual shared memory environments as the solution (Schel 1991).

The ControlCalc MIF solution solves the DCS performance problem by using a distributed spreadsheet matrix for data storage in shared memory, rather than a file system. The bandwidth limitation then becomes network and synchronization overhead. Distributed shared memory systems such as Encore's Reflective Memory System (see section 7.1.2. below) and high performance networks such as FDDI can solve network performance issues, as long as data is organized in a simple, memory based spreadsheet structure. The general-purpose MIF approach also solves the interoperability and limited domain problem.

3.2.6. Graphical Design and User Interface

While a spreadsheet improves ease of use and programming productivity, graphical interfaces must be presented to both system designers and end-users for maximum usability. A large number of function block diagram systems are currently available in the commercial market, such as LabView from National Instruments and VEE (Virtual Engineering Environment) from Hewlett-Packard. In these systems, a system is designed by graphically wiring together standard function blocks to create data-flow diagrams. As with DCS, function block diagram systems suffer from a number of problems: poor performance, limited domain, no distributed processing, and proprietary design. Performance is limited by the interpreted evaluation modes and message passing rather than shared memory data flow paths. Their domain is limited to data acquisition and analysis with no open interface for distributed or modular extensions.

However, the basic idea of data flow diagrams is a useful concept, when applied to an underlying engine that meets the performance and modularity requirements of ONR. Modular Interconnection Frameworks can be naturally expressed in graphic diagrams.

The major opportunity for the ControlCalc MIF system is to extend the diagram concept into the spreadsheet itself. The DCS-II proposal (see section 5.3.) makes each page, or task, of the spreadsheet a module on the Polylith bus. In addition, a task itself can be expressed as a set of function blocks with connections specifying the external references. Within a spreadsheet task, the connections would resolve to direct shared memory references and the function blocks would be implemented as sub-ranges or blocks of spreadsheet expressions. In addition to data flow connections, the diagram system will allow the designer to specify the order of evaluation and apply iteration control function blocks.

3.2.6.1. Driving GUI Interfaces with Spreadsheet Data

The current ControlCalc product provides a graphical user interface construction kit, including a GUI editor, a library of common control panel gadgets and an attachment declaration utility. This system reduces the problem of driving graphical interfaces to a mapping between graphical attributes and spreadsheet cells. The run-time GUI control task, like the spreadsheet user-interface task, monitors attached spreadsheet cell locations and redraw gadgets as required. Input gadgets can directly change spreadsheet setpoint cells, and can trigger command cells which, among other actions, can issue

commands that change window display attributes. This declarative approach has proven easy to understand and implement among ControlCalc users.

The opportunity presented by the MIF system is to enchance the spreadsheet GUI mapping system to support distributed graphics and modular, hierarchical gadget construction. Distributed graphics is another case of distributed spreadsheet data, and will be solved with the MIF interface. When the attached data cells are not local to the graphics system, they will either be attached in shared memory or through the Polylith message-passing interface, depending on the interconnection medium available. Hierarchical gadget construction will be done by using an object-oriented graphics package, and letting attachments at each object level be inherited.

The commercial potential of such a graphical interface is enormous. The ability to construct graphical objects and function blocks within a spreadsheet and to generate compiled runtime code to implement that diagram is simply not available today. Response from ControlCalc customers and users of competitive systems point to general market interest in higher performance and greater extensibility in graphical interface and design tools than current products provide.

4. Phase II Technical Objectives

As the information in this section includes details of customer applications and internal RTware product development plans, the entirety of section four is considered proprietary information.

4.1. Summary

The basic technical objective is to integrate the ControlCalc real-time spreadsheet into the Polylith MIF system, and provide a graphical interface for both system design and operator interface. ControlCalc's compiling, multi-tasking, shared-memory programming model will be extended to allow Polylith message-based distributed services to be declared that directly access the spreadsheet in the DCS-I stage. Tightly-coupled multi-processing with the multi-tasking spreadsheet model will be implemented in the DCS-II stage. Performance analysis and schedulability constraints will added through an interface to the PERTS system. Finally, graphical user interfaces will be developed based on the existing ControlCalc X-Windows graphical toolkit. The Phase II optional section will be the implementation of a modular function block diagramming graphical editor.

The spreadsheet package will be ControlCalc from RTware. Uniquely among spreadsheets, ControlCalc evaluates compiled code in a multi-tasking, shared memory system organized in a three-dimensional spreadsheet format. ControlCalc is already in use in demanding commercial and DoD applications such as turbomachinery control, mainframe monitoring, and airfield control tower control.

The MIF package will be Polylith from the University of Maryland. Developed under DoD contracts, Polylith is a software bus that supports network message passing and high level interface definitions between software modules running on heterogeneous computer systems. Language independence and a module interface language and compiler allow spreadsheet modules to communicate with modules implemented in languages such as C which have a Polylith interface. Easily portable, Polylith is in use already in distributed applications.

The Phase II effort will achieve the following technical objectives, with approximate percent of effort:

1. DCS-I 20%
    a. Remote access to the spreadsheet from the Polylith bus.
    b. Calling remote functions from spreadsheet expressions.
    c. Demonstration installations with customer applications
        i. Naval air field control towers

ii. UNIX system health and performance monitor

2. DCS-II 45%
   a. Distributed, multi-processing execution of spreadsheet logic.
   b. Porting to DSP sub-systems.
   c. Demonstration installation with the Naval Research Laboratory Orion project.

4. Performance Analysis: 5%
   a. PERTS interface.

3. Graphical User Interfaces: 30%
   a. Hierarchical graphical tool for run-time displays. (10%)
   b. Modular diagram system (20%) - This is the Phase II optional section.
      i. MIF network diagram
      ii. Spreadsheet function block diagrams

As the time estimates indicate, the DCS-II effort is the primary work of the project. DCS-I is less demanding and has been started in phase II. Two demonstration projects need only DCS-I and will be started when DCS-I is ready for initial release. The graphical interface will be development concurrently with DCS-II after the completion of DCS-I.

The following sections details each of these technical objectives.

4.2. DCS-I

4.2.1. Remote Access to the Spreadsheet from the Polylith Bus

The first objective is to allow external processes running on remote processors to access the spreadsheet using the standard Polylith Bus interface. This will be accomplished by allowing the spreadsheet application designer to use an interactive tool for declaring external interfaces in a functional form. Once a functional interface is defined, the tool will generate the necessary interface language files and optionally coordinate the initialization of the Polylith Bus. Access routines will include the ability to import and export data from the spreadsheet, to invoke execution of spreadsheet logic on behalf of the caller, and to apply various synchronization options to the operations requested.

Our Phase I work has resulted in a detailed specification of this tool (see the Work Plan, section 4), and preliminary work indicates implementation as designed is feasible. This objective is part of the DCS-I specification.

4.2.2. Calling Remote Functions with Spreadsheet Expressions

The second objective is to allow remote function calls from within spreadsheet logic. This means that Polylith function declarations will be made visible to the ControlCalc compiler for binding when the system is started. This objective is part of the DCS-I specification.

Phase I research showed that Polylith data types and function interface declarations are a large subset of ControlCalc types and interfaces. This means that all Polylith semantics will be supported, but certain ControlCalc semantics will not. However, the capabilities in question, involving fifo data structures, can be handled through a series of lower-level calls, and by provision for a specific fifo I/O transfer mechanism. The fact that the ControlCalc spreadsheet is compiled means that use of an external function definition is easily done. The natural requirement is that all functions be present and declared at compile time, with only warnings be posted during editing.

4.2.3. Demonstration Projects

Two demonstration projects are planned using DCS-I software. Both projects build on applications already designed using ControlCalc.

The first project is an extension of the Navelex air field lighting control system developed by Booz Allen Hamilton in Charleston, SC. That system is planned to be enhanced to add new functions that have to communicate with a variety of computers, including data acquisition systems, the collision avoidance computer system and weather data computers. Also, the system will include multiple operator displays, one at each operator station, each of which should be capable of displaying any of the information available through any of the interfaces. DCS-I technology will be used to import and export data between multiple spreadsheet applications and non-spreadsheet applications running on the various sub-systems. The system will be heterogeneous, with both different computer processors and different operating systems, and with mixed languages, particularly ControlCalc and C.

The second project is the generalization of the Encore CommandCenter system currently shipping on all Encore Infinity parallel-processing UNIX mainframes. Encore plans a joint development effort with RTware to develop a similar system for the high-performance UNIX data center environment. Different computer architectures and networks will have to be supported. Most networks will be message passing (typically TCP/IP) rather than shared memory, so the general capabilities of a MIF will be an absolute requirement. The system will also have to be able to communicate with existing programs, written in a variety of languages, which already handle particular system management functions.

Both demonstration projects will require DCS-I, although some features of DCS-II will be useful. Both organizations involved have supplied supporting statements for this proposal (see section 8.5.).

4.3. DCS-II

4.3.1. Distributed, or Multi-Processing, Execution of Spreadsheet Logic

Distributed or multi-processing execution of spreadsheet logic is the major technical objective of DCS-II. The distinction is terminology we use here reflect loosely coupled versus tightly coupled processors. There are a number of specific technical advantages gained in general by multi-processing, especially performance and by distributed processing, especially robustness. In addition, there is a major technical objective in real-time systems that is achieved through multi-processing, that is "hard" real-time distributed.

High speed, or "hard" real-time applications require an operating system kernel designed for such applications, including features like direct physical addressing, rate-monotonic scheduling, etc. There are many such kernels commercially available, and ControlCalc, for example, runs under one. However, the requirements made on these kernels conflict with the requirements for an operating system used for multi-user programming, editing and graphical user interface. The standard, open-systems platforms that are successful today (POSIX, Windows-NT, etc.) are specifically not capable of hard real-time processing. The solution is to split an application into two components: real-time processing and non-real-time processing (or "soft" real-time). Typically the "soft", or less time-critical, components involves user interface, data logging, etc. The editing, configuration and programming of an application also falls into the soft category.

The distinction between soft and hard real-time processing leads to a natural modularization of an application. The objective of DCS-II is to allow spreadsheet application designers to specify that any spreadsheet task is to run on an attached processor. The compiler will generate the necessary type of remote data calls.

4.3.2. Porting to a Selected Set of Platforms

The Phase II project will include porting to a set of platforms including real-time operating systems, open-systems operating systems and various process targets. The objective is to have a full set of options which are appropriate for commercial and DoD use. The proposed platforms are:

a) Solaris/Sparc
b) LynxOS on 486 and 68040 processors
    LynxOS is a real-time UNIX system, approved by DoD.
c) IXTHOS DSP VME-bus sub-system running SPOX.
    This system is specified in the NRL project RTware is working on.
d) National Instruments DSP processor cards running SPOX.
    These cards run the same real-time low-level O/S as the IXTHOS system, but go into PC-AT machines.
e) VMEexec running on Motorola 68040 VME systems.

ControlCalc and Polylith have already been shown to be easily portable to UNIX and UNIX-like operating systems. The major work required to achieve this objective is porting the ControlCalc code generator to the Sparc processor and to the DSP processors. ControlCalc has now been ported to three processors families (486, 68xxx, 88xxx) including a RISC processor, so a realistic time frame of 4-6 months per family is known.

The port to DSP processors is a major technical goal of the DCS-II project. This is for business as well as technical reasons. Technically, the DSP sub-systems provide extremely high processing for data acquisition and floating point computation, and are capable of stand-alone feedback control operation, with both analog input and output. Furthermore, programming such systems must now be done entirely with low-level C programming requiring a detailed understanding of the DSP hardware and low-level operating system kernel. The technical "fit" between the spreadsheet using fifo history cells with array operations and the signal processing routines of the DSP board libraries is very close. The tightly coupled multi-tasking required for high-performance DSP applications also fits very well in the shared memory multi-tasking model of ControlCalc. From a business perspective, providing the ability to program DSP systems through a spreadsheet without any loss of performance will be a breakthrougl, in a market which is already rapidly growing.

4.3.3. DCS-II Demonstration Project

The demonstration project for DCS-II will be radar data acquisition, processing and display for the Orion aircraft radar systems, through the Naval Research Laboratory. ControlCalc has already been selected for the graphical interface and system control part of the project. The IXTHOS DSP card has also been selected for this project. While the project is just starting now, and the final design is not complete, the principle engineers involved have indicated that this ONR MIF proposal fits in exactly with their long-term design goals.

4.4. Integration of a Real-Time Scheduling Analysis Tool

The Phase II project will include use of parts of the PERTS system from the University of Illinois to allow users to profile and test the schedulability of real-time applications. This work was done under a DoD contract. Based on Dr. Jane Liu's presentation at the May '93 Prototech conference, it is feasible to export the scheduling information and performance constraints from a ControlCalc application and use the analysis tools of PERTS to determine the limits of real-time behavior.

This objective is self-contained and does not directly impact the main objectives of developing the DCS-I and DCS-II systems.

4.5. Graphical Interfaces

Two technical goals focus on graphics: run-time displays for operator interface for the application itself and graphics diagram editing to improve software productivity in application development. The graphical diagram goal is presented as the Phase II optional portion.

4.5.1. Hierarchical Graphical Tool for Run-Time Displays

The run-time display goal is to build on the existing graphical display capabilities of ControlCalc to add hierarchical graphical structure and improve support for distributed graphic displays.

Currently, ControlCalc has support for an internally-developed graphics package running under X-Windows for UNIX systems and a similar third-party package running under G-Windows for OS-9 systems. Support for the commercial DataViews product is currently under development. All the graphical interfaces support multiple logins to a running spreadsheet application and can support multiple hardware display systems. However, in both cases there must be a separate spreadsheet process controlling each display and running on the target system.

The first technical goal is to improve graphics distribution by allowing the mapping between spreadsheet cells and graphical objects to be supported by general-purpose query functions issued against a spreadsheet. There will be a query server which would accept requests from display processes over the Polylith bus, look up or update data in the spreadsheet and send responses. This would allow, for example, multiple spreadsheet instances to be mapped into single displays, which cannot currently be done. A set of cooperating modules could therefore present a unified operator interface. Display screens could be designed which consolidate information from a variety of modules, and allow the operator to navigate among module displays within a seamless graphical environment.

The second technical goal is to add support for hierarchical, or object-oriented, graphics. Currently the graphics attachment process is flat, so the user cannot easily create libraries of complex graphical gadgets built out of simpler gadgets. Both the graphical editors and the spreadsheet attachment structure will be changed to support hierarchical structure with attribute inheritance.

4.5.2. Modular diagram system

The first objective in this section is to construct a graphical diagram editing tool for visualizing modular interconnections. This tool will basically allow the Module Interconnection Language files to be generated by parsing a directed graph diagram of the module interconnections. The tool will include visual objects which indicate the synchronization primitives applied to each interconnection. Run-time monitoring of interconnection traffic will also be provided.

The second goal of this section is to provide a function block, or data-flow, diagram front end to the ControlCalc spreadsheet. This will allow the blocks of spreadsheet functions, data cells, expressions, an icon and associated graphics to be declared a function block object and stored in an function block library. A graphical diagram editor will be used to design an application by inserting and connecting function block icons. The connections will be used to resolve external references from each function block, and to indicate connections between fifo buffers and signal sources and sinks. Order of evaluation constraints will be settable, as will such attributes as locked, synchronized evaluation.

The locked evaluation option is an important goal as it allows the ControlCalc compiler to completely eliminate the overhead associated with existing data-flow diagram systems (e.g. LabView from National Instruments and Virtual Engineering Environment from Hewlett-Packard). In existing systems, function blocks are data driven, in that they execute only when data is ready. This means that each block must be

individually schedulable by some data flow monitor, often implemented as operating system pipes. However, a typical control or acquisition application in fact works in perfect lock-step, where each data point is operated on once by each function block in a predefined order. So while feedback control algorithms are often expressed in diagrams, the existing data-flow runtime systems provide extremely poor performance when used for control systems. With the ControlCalc compiler able to resolve such synchronized data sets to direct shared memory references, the technical goal of this portion is:

To provide a data-flow diagramming system which maintains the very large performance advantage that ControlCalc currently holds over existing commercial systems in feedback control algorithms.

5. Phase II Work Plan

All Phase II work will be performed at RTware's Durham, North Carolina facility. All the work is software engineering, and will be performed on in-house computer systems. At least 90% of the work will be performed by RTware personnel, either direct employees or full-time contractors. Additional consulting work with the Polylith and PERTS software packages will be performed by qualified personnel at organizations directly involved with these products.

All the following information in this section (section 5) is proprietary information.

5.1. Summary

The Phase II effort is planned over a 24 month period. Each stage of work is enumerated and described in the following table. All times are expressed in months starting from the beginning of work. Milestones describe events within a stage, and are also expressed in months from the beginning of work on the entire project. Total effort is expressed in engineering-months and reflects the amount of engineering time estimate for the entire stage. The total effort figures are used in section 13 as the basis of the cost proposal. Detailed descriptions of the work plan for each phase are presented in following sections. The detailed descriptions do not provide further time estimates, instead they describe in as much detail as is practical the specification of each phase of development. Note that multiple personnel will be involved and multiple stages will be in progress simultaneously.

As this phase II proposal involves primarily development of concepts and specifications developed in phase I, the specification describes the work to be done. Note that the DCS-I and graphical spreadsheet programming tool specifications have been worked out in great detail, and implementation was started in phase I. In the interests of readability, the specifications of these stages have been placed at the end of this section so that the stages with more technical interest can be read first

This technical work plan includes early commercialization of DCS-I after twelve months of effort, including demonstration and beta testing in ongoing ControlCalc applications. Section eight presents a more detailed commercialization plan. Note that each of the technical stages of this effort involves significant enhancements to RTware's software product set. Commercial interest by existing and potential RTware customers is expected to be very high, allowing technical features to be reviewed and revised through on-going customer interaction throughout the design and implementation effort.

5.2. Stages of Work, Milestones and Estimated Effort

Stage	Description	Start Time	Milestones	Total Effort
1. DCS-I	Remote spreadsheet access	month 0		16 months
	Milestones:			
	a) Specification complete		month 0 (completed in phase I)	
	b) Communication Interface Running		month 1 (partially done in phase I)	
	c) Interface Declaration Tool:		month 2	
	d) Compiler References to external functions:		month 4	
	e) Demonstrable use in demo projects:		month 6	
	f) Commercial availability		month 8	
2. DCS-II	Distributed Spreadsheet Operation	month 2		40 months
	Milestones:			
	a) Detailed Specification		month 4	
	b) Stand-alone run-time		month 8	

	c) Compiler-generated interfacing	month 14	
	d) Code Generation for DSP processor	month 17	
	e) demonstration at NRL	month 20	
	f) DSP commercial availability	month 22	
	f) Additional code generators	month 24	
3. PERTS	Performance analysis and schedulability	month 18	4 months
	Milestones:		
	a) Exporting Data to PERTS	month 20	
	b) On-line PERTS access	month 22	
	c) Commercial availability	month 24	
4. GUI	Graphical User Interface	month 0	24 months
	Milestones:		
	a) DataViews sub-view support	month 2	
	b) Inheritance specification	month 4	
	b) Inherited spreadsheet interface	month 10	
	c) GUI server through Polylith	month 16	
	d) Function Block Diagram Specification	month 20	
	e) Commercial Availability	month 24	
5. GUI Option	Interconnection and Function Block Diagrams	month 14	20 months
	a) Diagram Specification	month 16	
	b) Commercial toolset selection	month 17	
	c) Diagram Edit Tool	month 20	
	d) Spreadsheet Generation	month 22	
	e) Run-time visualization	month 24	

5.3. Distributed Control System II: Distributed Code

5.3.1. Exploration of DCS-II Issues

DCS-II is defined as the transparent distribution of ControlCalc compiled control "scans" to slave processors. ControlCalc is already a multi-tasking system, where each page of the three-dimensional spreadsheet evaluates in a real-time compiled mode. Each scan is controlled by a separate process, with its own trigger mode (cyclic, event-driven, interrupt service). The compiled code directly references the shared memory spreadsheet data structure, allowing cells calculated in one scan to be referenced by another at any time. Counting semaphore functions allow synchronization between scans, through, for example, the construction of monitors. Although scans are implemented as processes, the system is conceptually a threaded system, where the spreadsheet data area is the common data space of a process and each scan is a thread of control running within that data space. In order to understand the issues involved in DCS-II, it is useful to review the mechanisms underlying ControlCalc's real-time scans.

5.3.1.1. The Compiled Spreadsheet Structure

The basic mechanism is to compile the formulae entered in the spreadsheet into C-callable subroutines, one for each page of the spreadsheet. Natural spreadsheet evaluation order (row-major) provides the basic flow of the program. GOTO and GOSUB functions allow programming branches, loops and subroutines, similar to a BASIC program. Unlike BASIC, however, the normal result of each statement is to compute and return a value which becomes the current value of the cell containing the statement. In a very real sense, this type of spreadsheet evaluation can be considered a rule-solving system, where sets of rules (logical, text or arithmetic functions) are scanned to recompute the state of a system. This has a strong correspondence to control systems programming, which is often considered a set of control laws

that are applied to a set of inputs in order to generate a set of outputs. Many functions, of course, operate primarily through side-effects, which we will consider below. One important feature: the spreadsheet does not allow address computation, except through indexing operations into ranges, or tables of data in the spreadsheet.

Disregarding side effects, the rule-solving evaluation mode results in a simple model of distributed evaluation, based on each scan being assigned to a separate, remote or tightly coupled processor. In downloading a page of the spreadsheet to a remote processor for scanning, one is also downloading the data space associated with all writes by that page, since a function's data and formula (or rule) are two attributes of a spreadsheet expression cell. Since no address computation is allowed on writes, inter-page references can be captured at compile time and packaged for remote access. In ControlCalc, range references are restricted to intra-page only. That means that ranges can only be two-dimensional and must therefore be fully contained within one page.

5.3.1.2. Handling Side-Effects

There is one important class of side-effect functions which violate the rule-solving nature of the spreadsheet. These are the "SETVAL" functions. SETVAL and its indexing cousin ISETVAL are the ControlCalc names for the functions which can write a value into a cell other than the current expression cell. These are assignment functions. There is no restriction on page locality for these functions. However, like all cell references, they either reference to a specific cell, determinable at compile time, or they index into a specified range of cells, determinable and restricted to one page. These functions are included in the ControlCalc programming model to simplify programming of state systems. Without these, a spreadsheet is a stateless language.

While stateless languages are quite robust, and easily understood for stateless applications, they require excessive complexity when trying to implement algorithms that are essentially state machines. In control programming, a combination of state and stateless programming is usually required. Stateless programming is used to apply invariant rules to a system. Continuous feedback algorithms are usually stateless, using running integrators and differentiators to measure time-dependent behavior. Safety interlocks are best implemented in a stateless manner. To do that, all output devices should be written at one point in the process. That way, illegal or unsafe conditions can be checked to minimize problems caused by, for example, unforeseen errors in a state sequence. On the other hand, many real-world processes are state machines, involving a sequence of events driven by changing real-world inputs. Implementing these sequences as a set of invariant rules makes the rules quite complicated, while a state sequence implementation would be very simple. Therefore a complete control programming system must include both capabilities. The impact of assignment functions on a distributed processing system is to require support for remote writes as well as reads.

5.3.1.3. Packaging Remote Data Access

In packaging remote data access, there are two choices: message passing or data mirroring. Message passing means that every time a remote data point is referenced within an expression, the compiler generates a remote read call directly in-line in the code. Data mirroring means that the remote data set is block-transferred at some point and then accessed locally. While data mirroring is more efficient, it cannot be implemented reliably without a caching mechanism. Such caching would mean that data would be transferred through to remote nodes when written (write-through cache) or only when changed and required by a read (deferred write-through). The overhead of doing a software cache over a network makes it impractical. However, on systems with shared memory hardware, it is very practical. ControlCalc currently runs on such a system: the Reflective Memory System (RMS) used by Encore's Series 91 and Infinity computers.

In the absence of distributed shared memory hardware, we intend to implement a full message-passing system. This approach can achieve the performance of a data mirroring system, if the application was designed with distribution in mind. ControlCalc provides block copy functions (another variation on SETVAL) which lets data be copied between different ranges of the spreadsheet. By setting of the system so that remote data is block-transferred precisely when needed, local references can made to the copied data, greatly reducing the number of remote accesses required.

The explicit block transfer approach fits well with stateless programming techniques. In fact it can be an advantage in the software design because data references are packaged and executed only at one point in the logic, eliminating the possible race condition bus that often occur in multiprocessing. The disadvantage of explicit block transfers is that they must be explicit. The application therefore becomes less portable. More precisely, it becomes potentially less efficient when configured on a shared memory or single-processor system.

Message passing can be considered in two forms: explicit and transparent. Explicit message passing would involve a message from one scan requiring a response from another. This mechanism is already provided using the remote procedure calls running over Polylith. Explicit message passing requires the use of specific remote access functions.

A transparent message passing system would involve a data server process running automatically on each node that is running ControlCalc scans. The compiler would generate a standard message call that would request data from a remote node whenever a remote reference is encountered in the program. Since all scans are compiled at once, such data references could be pre-compiled down to indexed table lookups, reducing the overhead of traversing the spreadsheet sparse matrix to find each data point. The transparent system would also be implemented with Polylith, but without requiring specification by the user.

Finally, in all implementations, it is necessary that counting semaphores be supported across the entire distributed environment. Without this capability, it is difficult or impossible to create tightly coupled applications that work reliably in distributed environments. Once again, ControlCalc does not allow signal address (or i.d.) computation, so the compiler can determine the set of scans that are attaching to specific signals. Signals can therefore be arbitrated by any one of the set of scans, without requiring a central signal arbitrator. This will increase the robustness and performance of the system by allowing signal communication within multiple localized areas of the distributed application.

The key to efficient programming with transparent message passing is for the programmers to understand the mechanism and the cost of remote access. In particular, they should be aware that block transferring data prior to using it, and synchronizing block transfers with signals will both minimize communications traffic and increase software reliability.

5.3.1.4. Advantages of DCS-II

The advantage of DCS-II is that no programming is required to explicitly pass messages between the different scans of the application. The ControlCalc spreadsheet programmer already interactively declares a scans operating mode, from a short list of choices and parameters. To run a scan remotely, only some additional parameters will be required in that declaration. These parameters would specify the host, the network port with any associated configuration parameters and possibly the network operating mode.

One of the key design goals is to preserve the fundamental advantages of the ControlCalc spreadsheet paradigm. In particular, the system must present on-line updates from all active nodes to the user-interface part of the spreadsheet. Right now, ControlCalc users can watch spreadsheet cell values changing in any scan in the system by simply looking at the cells which define the function being

executed. This is inherent in the shared memory design of the spreadsheet, since both the executing scans and the spreadsheet interface are using the same shared memory locations for the data. The user interface program is simply polling and displaying the most recent result at a speed (about 5 Hz) which is adequate for human reaction. This shared memory approach also allows the inclusion of custom graphical user interfaces by mapping the variables associated with graphics objects into the same shared memory areas. Manipulating graphical objects is thus done through a declarative interface which establishes bindings, and does not require any programmatic language support for the most common operations.

5.3.1.5. Implicit Message Passing

The most difficult part of the DCS-II design is the implicit message passing between the "threads" of a ControlCalc spreadsheet. ControlCalc's organization allows the user to declare any page of the three-dimensional spreadsheet to be a scan ("thread"). Each page can contain both function cells with both the expression and its result and data cells of various types. This organization lends itself well to the DCS system because the data on a page can be kept locally on a remote note along with the code for that page.

However, the spreadsheet model treats all data cells as globals in the sense that cells on one page can be read by expression cells on another. There are even cell write functions that do cell assignments by side effect. Note that except for the cell write functions, a spreadsheet expression writes its result to the data part of the expression cell itself, so by far the majority of writes are local.

The spreadsheet compiler will have to handle remote reads and writes, and should do so without requiring any explicit programming. The simplest method is to have the compiler convert all remote reads and writes to message calls, in-line at the point the executing function requires the access. This will maintain complete consistency when done with a bus such as Polylith that insures atomicity of these operations. The question is whether perform can enhanced using caching schemes.

A write-through cache approach would require each remote node to maintain a cache table of remote cells that are referenced by local code. This is easily done with the compiler, since scan location is declared before code is generated. There would be the overhead of locking the cache before reading the value, and writes would always involve message calls, but we anticipate a major reduction in message call traffic with this approach.

More sophisticated caching schemes may not be required for one simple reason: ControlCalc provides block copy functions which can operate between pages. This means that the programmer can optimize the application by understanding that it is much more efficient to explicitly move blocks of data between remote pages than to rely on the implicit message calling.

5.3.1.6. Target Platforms

A target platform for the DCS system impacts ControlCalc in 1) the cpu architecture, which requires a cross compiler, 2) the operating system and 3) the network interface. To this point, Polylith has focused on UNIX platform with TCP/IP networks. The C compilers are, of course, already available. In the real-time control area, however, platforms are much more heterogeneous.
The second phase of a ControlCalc-based DCS system will allow the code segment processes (known as scans) to operate on distributed and heterogeneous hosts. This is complicated by the fact that ControlCalc scans more closely resemble threads than processes. As such, they could be distributed across processors on a tightly coupled multi-processor platform running, for example, Mach. However, a DCS system is usually loosely coupled with a LAN network. For this reason, intelligent compiler technology is required to minimize the network traffic required to maintain coherence between distributed copies of the shared memory data space.

5.3.1.6.1. Digital Signal Processing

DSP processor are greatly increasing in popularity for real-time applications, not just for classical signal processing but for their high-speed data acquisition and raw processing power. The NRL group is planning to use a DSP processor on a VME board from IXTHOS Corp. That board uses the ADSP-21020 processor from Analog Devices, which can achieve 100 peak and 66 sustained MFLOPS. This system also comes with a set of mezzanine I/O modules with, for example, a 12-bit, 10 MHz A/D converter. The operating system with that board is SPOX, which is also widely available on commercial DSP processors such as those from National Instruments on PC-AT systems. SPOX is not UNIX, nor does it support TCP/IP for communication with the host computer. However, it does have a rich set of message passing functions, making it is feasible to port a version of Polylith to SPOX. The Phase II project will use the IXTHOS system as a target processor for the DCS_II objective.

5.3.1.6.2. Real-Time Unix

In the general-purpose processor market, there are a number of popular real-time operating systems which have various levels of compatibility with UNIX and TCP/IP. ControlCalc currently uses the OS-9 system from Microware which supports the socket library. OS-9 is self-hosted with no multi-processing capabilities. U.S. government agencies, including DoD and NASA have worked extensively with LynxOS, VxWorks and VMEexec. Of these, LynxOS is closest to a real-time, self-hosted UNIX, while VxWorks and VMEexec are designed with remote real-time kernels that use UNIX for hosted development and support the full TCP/IP protocol. The NRL project is evaluating LynxOS, due to its high POSIX conformance and self-hosted real-time capabilities. The DCS-II objective includes porting to LynxOS and VMEexec. VMEexec was selected due to the relationship with Motorola Corporation and RTware, and its widely available and high performance real-time operating system kernel (PSOS from Software Components Group).

5.3.1.6.3. Standard UNIX

Finally, a DCS application will include systems which are not real-time, particularly for functions like operator interface, configuration control and back-end processing. For these systems, the standard is UNIX running on RISC workstations. Much of the Phase II work will be done on Sun Sparc systems, since Sparc is a de facto industry standard. Due to our important relationship with Encore Computer Corporation, the system will also continue to run on Encore's Series 91 and Infinity computers, based on the 88K RISC processor.

5.3.1.6.4. Cross Compilation

In order to run on heterogeneous targets, ControlCalc's internal compiler must be capable of generating code as a cross compiler. Currently there are three versions of the compiler, generating code for the 386/486, 68K and 88K processors. However, the compiler is not packaged separately, so it will have to be repackaged and supplied as a set of separate cross compilers. This work is included in the Phase II, DCS-II objective.

5.4. Integration of a Real-Time Scheduling Analysis Tool

The Phase II project will include use of parts of the PERTS system from the University of Illinois to allow users to profile and test the schedulability of real-time applications. This work was done under a DoD contract. Based on Dr. Jane Liu's work and the system documentation provided, it is feasible to export the scheduling information and performance constraints from a ControlCalc application and use the analysis tools of PERTS to determine the limits of real-time behavior.

This work will not involve porting PERTS to any other platform except those it is already available on. It is expected to be used primarily on Sun systems. Porting to other systems will be considered after commercialization and motivated by market demand.

5.5. Graphical Interfaces

Two technical goals focus on graphics: run-time displays for operator interface for the application itself and graphics diagram editing to improve software productivity in application development. The graphical diagram goal is presented as the Phase II optional portion.

5.5.1. Hierarchical Graphical Tool for Run-Time Displays

This phase of the work plan involves adding support for hierarchical, or object-oriented, graphics. Currently the graphics attachment process is flat, so the user cannot easily create libraries of complex graphical gadgets built out of simpler gadgets. Both the graphical editors and the spreadsheet attachment structure will be changed to support hierarchical structure with attribute inheritance.

Note that the term gadgets is used to refer to graphical objects whose attributes can be directly controlled by mapping them into spreadsheet cells or tables. Gadgets are control-panel oriented, and include such devices as:

1) X-Y Plots
2) Strip Charts and Trend Charts
3) Bar Charts
4) Needle Gauges
5) Input Sliders
6) Digital and Text Readouts
7) Numeric and Text Entry boxes
8) Pushbuttons
9) Moving images
10) Hot Spots
11) LED-indicators
12) Geometric Shapes
13) Radio Buttons

5.5.1.1. Upgrades to the Editors

RTware's current X-Windows editor will be enhanced to add a container gadget for grouping collections of lower-level gadgets, including other containers. Containers will be able to be saved individually to files to create libraries of complex graphical objects. As in the low-level gadgets, multiple named instances will be allowed. Work will also proceed on the DataViews product interface already in progress, making use of its "sub-view" capabilities for similar functionality.

5.5.1.2. Inherited Attachments

ControlCalc drives graphics allowing the user to specify attachments which map spreadsheet cells to graphical gadgets attributes. In enhancing this approach to support complex container gadgets, it is necessary to allow sets of attributes to map into tables of data. The user will be able to define a prototype table in the spreadsheet and attach individual graphical attributes to cells in the table. The table's layout will be user-determined. The prototype table will be stored with the container gadget. When a container gadget is instantiated, the attachment instantiation will be done by simply selected a position for a copy of the prototype table. The standard spreadsheet features of relocatable cell addressing and range cut-

and-paste will be used to implement this. The prototype's contents then become the default attachment characteristics.

If a container gadget contains other container gadgets, then the higher level container gadget's prototype table will contain other prototype tables. It is necessary that changes can be made to the default prototype and be inherited even when the prototype is contained within a higher-level prototype. The container gadget will therefore reference the prototypes of sub-containers, in addition to keeping an instance, or local copy, of the prototype. This also requires a new cell attribute that will specify that a prototype's local copy cell value overrides the original prototype.

5.5.2. Client-Server Attachments

The second technical work phase will improve graphics distribution by allowing the mapping between spreadsheet cells and graphical objects to be supported by general-purpose query functions issued against a spreadsheet. Currently there must a separate spreadsheet process controlling each display and running on the target system. The Polylith interface developed for DCS-I will used as the basis for a "query server" which would accept requests from display processes over the Polylith bus, look up or update data in the spreadsheet and send responses. This would allow, for example, multiple spreadsheet instances to be mapped into single displays, which cannot currently be done. A set of cooperating modules could therefore present a unified operator interface. Display screens could be designed which consolidate information from a variety of modules, and allow the operator to navigate among module displays within a seamless graphical environment.

Note that this model differs from client-server model supported by X-Windows itself because the services are concerned with data sources for gadgets and can be per gadget, rather than just per-window graphics drawing information.

5.5.3. Modular Diagram System

There are two tools which will result from this stage of the Phase II effort. First is a function block diagram system as a front-end programming system for the spreadsheet. Second is a module interconnection diagram system for the Polylith bus.

5.5.3.1. Function Block Diagrams

This section of the phase II work plan will develop a function block, or data-flow, diagram front end to the ControlCalc spreadsheet. This will allow the blocks of spreadsheet functions, data cells, expressions, an icon and associated graphics to be declared a function block object and stored in an function block library. A graphical diagram editor will be used to design an application by inserting and connecting function block icons. The connections will be used to resolve external references from each function block, and to indicate connections between fifo buffers and signal sources and sinks. Order of evaluation constraints will be settable, as will such attributes as locked, synchronized evaluation.

The locked evaluation option is an important goal as it allows the ControlCalc compiler to completely eliminate the overhead associate with existing data-flow diagram systems (e.g. LabView from National Instruments and Virtual Engineering Environment from Hewlett-Packard). In existing systems, function blocks are data driven, in that they execute only when data is ready. This means that each block must be individually schedulable by some data flow monitor, often implemented as operating system pipes. However, a typical control or acquisition application in fact works in perfect lock-step, where each data point is operated on once by each function block in a predefined order. So while feedback control algorithms are often expressed in diagrams, the existing data-flow runtime systems provide extremely poor performance when used for control systems. Function block diagrams created in a spreadsheet and executing at compiled ControlCalc speeds are expected to have major commercial impact on

ControlCalc, based on customer request and the popularity of function block diagram in the controls market.

5.5.3.1.1. Range Files as Function Blocks

The fundamental function block design will be based on the existing "range file" capabilities of ControlCalc. This capability allows users to save any rectangular range of the spreadsheet to a file. The file contains all the cells in the range and the names of all cells in that range. The range can later be loaded into another spreadsheet. When loading, the user specifies the top left corner location of the range. The loaded range has the same shape as the original range. All relocatable cell references are adjusted to reflect the new location and the range's names are added to the spreadsheet's tag name table.

This simple function has a number of limitations that will need to be changed to support true function block programming.

5.5.3.1.2. Enhancements Required for Range Files

The existing range file capability allows only one copy of a range file to be loaded into a template if that range file contains any symbolic names. If the same range file is loaded again, there will be duplicate name conflicts. One solution to this problem is to load a range file by specifying a new location and a name for that instance of the range in the spreadsheet. By doing this, the internal tag names can be concatenated with the instance name to maintain uniqueness. This solution makes the naming private to each instance of the range. It would also be possible for external functions to reference these internal variables, but that would not normally be done within a function block system.

External references to the range file function blocks currently must be done manually. However, the relocatable nature of the block's cell references does allow some shortcuts. For example, the range file could have a reference to a cell one column to the left of the block in a particular row. When that range file is loaded, that reference would relocate and refer to whatever cell is in that relative location in the new spreadsheet. By doing this, and by avoiding the use of symbolic names in the range files, users currently can paste range files in repeatedly, and create tables of parameters in such relative locations. This method is not sufficient for a true function block system, which would require references between blocks to be resolved by the user drawing lines between attachment points on different blocks.

To support this dynamic linking between function blocks, we propose to add a new type of cell called a "link" cell. A link cell will be a cell which can contain only the name of another cell. It would be typed just like expression cells, and could be either empty or contain the text name of another cell. Expressions could reference link cells normally. When the spreadsheet is evaluated or compiled, references to link cells would resolve to references to the cell whose name is in the link cell. Of course, unresolved or improperly typed references would result in syntax errors. Note that link cells are needed only for external references from the range block. They provide a way of keeping all relocatable references inside the function block, and a place to record the resolution of the essentially indirect references. Link cells would involve no overhead in compiled run mode, since the link would be resolved at compile time. We do not envision a need for dynamically changing links while the system is running. A manual spreadsheet command would be available for resolving link cells by editing after the range block is loaded. In a function block diagram system, the link cells would correspond to input points on the block.

References into a function block from the outside can be accomplished by simply adding the attribute of external visibility to a cell. This would not actually be necessary if links were established manually within the spreadsheet. However, when range blocks are used in a function block diagram system, the creator of the function block needs to be able to specify which internal cells are visible externally. These externally visible cells would correspond to output points in a function block diagram.

5.5.3.1.3. Overview of usage

The system will be used in a two part process: creating function blocks and using a function block diagram editor.

Function blocks will be created inside the standard ControlCalc spreadsheet. The creator must follow the rule that all external references are made through link cells and must set the external visibility attribute where desired. The range can then be saved to a range file in the normal manner. A utility will be provided to allow the creator to specify an image piece and/or text name which will show up in the actual function block icons. Other decorative attributes of the block such as its shape and the position of the attachment points could also be specified with this utility. A default shape and position of the attachment points would be provided. Once created, the spreadsheet portion could be reloaded, edited and saved with its attributes.

Function blocks also have to be configurable. A set of parameters must be made available to the user, not as attachments, but for setting variables inside the block that are used by the expressions inside the block. These variables are entirely defined by the function block creator, and can be any type of cell. ControlCalc already has the ability to present a spreadsheet-based form(s) to the user in operator mode. This mode allows the user to cursor between pre-determined setpoint fields and change those fields. ControlCalc also supports the attachment of graphic windows to spreadsheet cells. These graphic panels can be used for changing or viewing data in the spreadsheet using a rich set of graphics gadgets. Either method could be used within the function block diagram system. In both cases, there are potentially two displays required: an edit-option display and a run-time parameter adjust display. The function block creator will be able to attach graphic windows to the block and indicate which window is shown initially under the two conditions. The macro command cell capability already allows windows to be dynamically called up and removed by the end user. The creator simply needs to indicate which macro cells are to be initially executed. The rest of the interaction is entirely determined by the function block's contents. If the simple spreadsheet interface is desired, a mode setting could be used to determine whether the indicated cells are macros to be executed, or the top left settings of a spreadsheet region to be displayed in operator mode. This determination could be made automatically by stating that if the cell is a macro cell, then it is executed, otherwise it is the top left corner.

Function blocks would be used with a function block diagram editor. This discussion will not cover the user-interface details of the graphical editor, as such systems are common and well understood. A general discussion and certain functional details are important, however.

The function block diagram user will place icons representing blocks in the diagram and connect reference points. Different symbols will represent the different types of reference points, and the editor will complain if illegal connections are made. The user would also be able to open up the function block and set parameters using either the attached window(s) or spreadsheet operator mode.

The function block diagram editor should be hierarchical, so that function icons can be grouped together dynamically and viewed as a single icon. Such icons would have to have a pre-determined look, but present a dynamic set of attachments. The icons in a group would then be edited on their own display, and attachments outside the group would be indicated by attaching a border region or decoration of that display area.

5.5.3.1.4. Sequence of Operations

The most difficult design area for the system is sequence of operations. Most function block diagram systems are really data flow diagrams (as in LabView). These systems schedule functions as data becomes available. This approach can result in unnecessary overhead and can also be non-deterministic, both of which are problems for real-time control systems. The data-driven approach requires that each block be separately scheduled and that an executive monitors data sources. While this is appropriate in some cases, in many applications the function blocks are best executed in a synchronous sequential

manner. This is particularly true in feedback control systems. In these systems, such functions as input, scaling, limit checking, feedback equations and output are all inherently synchronized and run at a rate specified by the user. The obvious case for data-driven evaluation is in message-passing or networked applications where data is processed, possibly asynchronously, only when a message or new data is available. Distributed control systems are the obvious example.

ControlCalc itself already allows both methods to be implemented. Multiple scheduled code segments are provided since each page of the spreadsheet is a separate control task. Synchronization is provided through the signaled scan option, and message passing or data piping through history (FIFO) cells. History cells can optionally be configured so that read accesses wait for data and write accesses wait for room, and with locking to ensure exclusive access. Monitor-style control can be constructed using the signal and wait primitives. A network-based I/O option also allows data to be piped between history cells on different machines using streams. On the other hand, within each page execution is strictly sequential (disregarding GOTO's). Depending on its requirements, an application can be as simple as a single scan doing one continuously iterating control loop or a whole set of cooperating scans scheduling themselves as necessary.

5.5.3.1.5. Sequential Evaluation

Within a function block diagram system some method has to be provided that lets the user take advantage of both types of scheduling. The simplest method is sequential evaluation. This would allow the user to specify that the entire diagram, or a group within the diagram is executed sequentially. Sequential evaluation would probably be the default mode. The system would use built-in rules based on examining connections to attempt to determine the proper sequence of evaluation. The user would be able to override those rules by drawing explicit ordering links between blocks. Circularities in the ordering links would be forbidden.

This results in a system which is static and sequential in its evaluation order. The underlying method for evaluating the function blocks would actually be the ControlCalc spreadsheet compiler. Each block would be placed in an arbitrary location in the spreadsheet, which then provides the means for recording the linkages the user creates. The compiler would compile the blocks not in the standard row-major order of the spreadsheet, but block by block according to their ordering. This would be an elaboration of the "one page is one task" structure now being used by ControlCalc. Instead, diagram information would be used to drive the compilation and actual spreadsheet layout would not be important.

5.5.3.1.6. Multi-tasking evaluation

The user would be able to define multiple groups that could be separately scheduled. References between groups would still be allowed. Currently, multiple tasks are allowed on one machine, with all pages sharing the three-dimensional spreadsheet as a sort of shared memory data base. This approach would still work with the diagrams. All groups would actually be placed in the global spreadsheet, so data references would resolve automatically. The compiler could also be used to handle distributed applications, where groups are evaluating on different machines. Inter-group references would compile to message-passing calls rather than simple data references.

5.5.3.1.7. Event or Data-Driven Evaluation

Event-driven evaluation is already supported by ControlCalc. Currently a page can be set up as a signaled scan, which means it evaluates once each time it receives a signal (event). A signal function is provided to allow spreadsheet expressions to send signals. Also, a wait function is provided to allow scans to synchronize at any point in the code. This capability is implemented using global events, either named events under OS-9/OS-9000 or System V IPC semaphores under UNIX.

Under a function block system, event scanning could be declared as an evaluation mode for a sequential group, instead of giving the group an iteration rate. This is equivalent to the method currently used with ControlCalc pages. Alternatively, a graphical method could be used, involving a standard function block that represents the global event. That icon could be connected to multiple signal senders and one receiver. Linking a cell to a signal send would require that references to that link cell use it as the signal i.d. number, and vice-versa. Sent signals would also be able to be linked to a schedulable group to trigger its evaluation, using a special attachment decoration on the icon. Timer functions could be used in this manner to graphically illustrate the source of iterative evaluation, since the difference between iterative and event-driven for the groups is simply one of how the evaluation is triggered. It is also possible to implement monitor functions (lock and unlock) which can provide exclusive control among any group of functions using the same monitor. The monitor icon would be slightly different from a signal icon, since it is initialized in a different state.

Finally, some enhancements can be made to the history cells to allow evaluation triggers to be set by a history cell reaching a certain threshold. This would require that functions that write to history cells check the threshold and send a signal when the threshold is exceeded. The history cell would have to maintain some state information which allowed another function to reset the signal so multiple signals are not sent for the same threshold condition. In general, there would have to be a history link cell which allowed a function block to externally reference a history buffer, but conceptually this would work the same way as simple data cells. A history cell configured to send signals on reaching a threshold would then result in an interface decoration on the function block that could be attached to a global event icon as a sender. With the locking option on history cells, this approach could be used to create completely data-driven systems. Using network drivers to stream history cells together would then allow a diagram to span distributed systems.

5.5.3.1.8. Graphical Operator Interface

While a function block diagram is a useful tool for a system designer, in production most applications should present the simplest possible control panel interface to the operator. ControlCalc already allows control panel graphics to be created and attached to cells in the spreadsheet. Each function block can be configured with its own control panel window for run-time use. The Phase II task will provide a way of creating a master control panel which can be attached to either the local function block control panels or to other global control panels for applications which don't need to involve the operator in the diagram system. The simplest solution to this requirement seems to be letting the user create a custom function block and link it to external data points in the usual manner. The ability to have a spreadsheet interface on-line for creating custom function blocks is crucial for this capability. It is likely that every application will have a semi-custom set of control panels, using the standard function block panels in some cases and not in others. It will also be useful to allow command macros to include sub-panels from function blocks, thereby allowing access to private data. This lets the operator, for example, select a PID control block and adjust its internal run-time parameters without having to export the entire parameter set from the block. Including these sub-panels would then allow windows with banks of sub-panels, similar to the modular design of hardware control panels throughout industry.

5.5.3.2. Module Interconnection Diagrams

This stage of the Phase II work plan will be to create a graphical editor and translator for interconnection diagrams. The tool will allow the Module Interconnection Language files to be generated by parsing a directed graph diagram of the module interconnections. The tool will include visual objects which indicate the synchronization primitives applied to each interconnection. Run-time monitoring of interconnection traffic will also be provided.

The product resulting from the Phase II effort will include an X-Window based graphical user interface option, as that is already a part of the ControlCalc product on UNIX systems. This subsystem is distinct

from a graphical interconnection diagram system in that it provides display and data entry into a running application. A diagramming system would be part of the application design process, although it could also be used at run-time to monitor system status.

Preliminary design of a diagramming system will focus on two software packages: DataViews from V.I. Corporation and MetaH from Honeywell. The DataViews system is already supported as a graphical user interface by DataViews running on the Encore UNIX machines. It is being given high consideration both due to its commercial availability on all major UNIX platforms and for technical reasons. Technically, DataViews includes a library of graphic routines that provide CAD-style diagramming support that are designed for creating such an editor. MetaH is a specific modular diagramming system with hierarchical structure that supports their real-time software development. The PERTS system also describes a task graph model of real-time systems. Implementation of the system will either use DataViews or enhancements to RTware's own graphical editor, as determined by the detailed design results.

5.6. Distributed Control System I: Remote Access - Functional Specification

5.6.1. Overview

The DCS-I functional specification describes two fundamental capabilities:

1) Presenting spreadsheet services and data to external processes.
2) Calling remote modules from within the spreadsheet.

Both of these capabilities can be implemented using the Polylith, and in fact make up the core of what Polylith does. There are two problems that have to be addressed, however. First, there must be a way of defining services, or interfaces, from within a ControlCalc spreadsheet. Second, there must be functions that let the spreadsheet programmers create calls to such services in their spreadsheet templates. Note that in ControlCalc a running spreadsheet template is a set of tasks processing the shared memory data space of a three-dimensional spreadsheet. Each task is a separate executing process, instantiating a set of spreadsheet functions. The spreadsheet function set is therefore considered a simple functional language.

To understand the requirements of the DCS system, it is useful to understand the general sequence of operations in the implementation of a distributed application with Polylith. Distributed Polylith applications are constructed in a modular fashion. Each module consists of an executable file and a MIL "module_description" section file. The module_description section describes a "service" as an executable program (source or binary) and the host it runs on, along with its object attributes and the Polylith interfaces the module will use (both input and output). Note that a node can declare multiple interfaces, and interfaces are declared as either input (sink or function) or output (source or client). An application consists of a set of module services instantiated by a master "bus" program that runs on each host involved. The bus programs requires that all the module_description MIL sections be linked together along with an application_description MIL section. The application_description section defines a node in the application, naming the node and specifying which module instantiates the node. Note that one module can be instantiated as multiple nodes, hence the declaration of node names mapped to module names. The application_description section also declares interface bindings by associating source/sink or client/function pairs. Each binding therefore maps two specific interfaces on two specific nodes. Once this is done, the bus program has all the routing information necessary for its operation. An application is actually executed by executing the bus program on each host, passing a parameter which is the linked MIL file. The bus program(s) then execute the module binaries, which can then use the interfaces.

Given this Polylith structure, any particular module can be created independently and used in multiple applications by simply rebinding its interfaces for each instantiation. This is the essence of a module interconnection framework.

5.6.2. Modular Interface to ControlCalc

Within the ControlCalc system, a module will be considered a particular sprcadsheet template. There will be two form of interfaces: standard and user-defined. Standard interfaces will allow remote processes to perform such generic operations such as stopping and starting the control logic. User-defined interfaces will allow remote processes to read and write information from the control system spreadsheet, and optionally to trigger, pend and wait using the spreadsheets counting semaphore primitives. These modular interfaces will be either function or sink interfaces in Polylith, since they exist solely to respond to external requests. However, since control scans are the user-programmed executable functions in the spreadsheet, the services invoked by triggering a scan can be programmed to perform any function the spreadsheet is capable of, including accessing subsequent remote services, as described below.

5.6.2.1. Module Implementation

The actual executable program implementing the template's interfaces will be named using the template file name with the extension .mod. This will be a 'C' function that will initialize by obtaining the interfaces it must support, and then loop continuously doing the Polylith bus function: mh_readselect. That function returns both a message buffer and a pointer to the name of the interface that the message came in on. The module program will then look up the corresponding definition for the message's interface. Each interface will have a list of spreadsheet data points which will be read or written. Interfaces will have a configuration option to synchronize with the real-time spreadsheet executing logic using the ControlCalc's existing counting semaphores. There will be a number of modes of synchronization, so that explicit coding of the sequences is not required:

1) Monitor r/w control:	Wait for signal A, read and write data, send signal A
2) Monitor reads:	Wait for signal A, read data, send signal A
3) Monitor writes:	Wait for signal B, write data, send signal B
4) Monitor service:	Wait for signal C, do service in list below, send signal C
4) Signal Service:	Read/Write data, send signal A
5) Wait Service:	Wait for signal A, read/write data
6) Request service:	Write data, send signal A, wait for signal B, read data.
7) Monitored Request:	Wait for signal A, write data, send signal B, wait for signal C, read data, send signal A

These modes (Signal service and Wait service in particular) can be also be used to synchronize external processes, treating the spreadsheet logic as a rendezvous mechanism. The Polylith interface will have no knowledge of this signaling involved, but reasonable naming of this type of interface can serve to document the purpose of services using these features. It is important in selecting these functions to be aware that some applications may use multiple instantiations of a module interfacing to one spreadsheet template. If this is anticipated, then it would be normal to use one of the monitor synchronization modes if interrupting an operation is not to be allowed. Note that the signals described abstractly as A, B, and C will actually be a configurable ControlCalc signal number. There are 500 signal numbers available for synchronizing tasks that perform operations on any one ControlCalc shared memory region. Since a template, and therefore the instantiations of the module interface, are associated with a shared memory region, these signals are common to all instantiations. This means that an interface with a monitor will allow only one instantiation to execute the monitored operation, but will not interfere with interfaces to another spreadsheet template, even though the same signal numbers may be used. On a UNIX system, this facility is implemented with SYS V IPC's, using a key number to allocate ranges of signal id numbers.

The ControlCalc spreadsheet is already a multi-tasking, shared memory system. Therefore the capabilities described above can be implemented quite easily. The spreadsheet data area already is kept in a shared memory segment as a sort of shared memory data base. A library of reentrant routines for reading and writing data, and sending and receiving signals, are already in place and used by programs such as I/O agents and GUI window agents that are a standard part of ControlCalc.

5.6.2.2. Data Types

Data types will be specified using the ControlCalc data types, which consist of either simple types (integers, booleans, floating point or strings) or arrays of simple types. An array type in the spreadsheet is a range, or rectangular section, of the spreadsheet. Structured types are not supported by the spreadsheet. There is therefore a corresponding data type in Polylith syntax for the data types in the spreadsheet. This will allow the generation of MIL language definitions directly from spreadsheet-style declarations. Other language interfaces can then access the MIL definitions without knowing about the spreadsheet data types (another design goal of MIF).

5.6.2.3. A Module Interface Definition Command in ControlCalc

In order to define these external interfaces, a module specification command will be added to ControlCalc's interactive command set. The module specification command will bring up a utility program that lets the user create, delete and modify interface definitions for the current template. These definitions will be saved as part of the template definition file. The user will also be able to cause a MIL file to be generated, and a new executable binary file for the module to be built. The MIL file will consist of a single "service" declaration, but without the host name field. The host name will be added by a later utility that builds the complete application. The command will be presented in an interactive window, presenting scrolling lists of input and output data points, radio button selectors for synchronization modes, text edit fields and push buttons for editing. The following commands/operations will be supplied:

Operations on a scrolling list of interfaces:

1) Add an interface.
2) Delete an interface.
3) Rename an interface.
4) Select an interface.

Operations on either of two scrolling list of data points:

1) Add
2) Delete
3) Edit
4) Verify data points

Mode Selections: Select one or none of the synchronization modes listed in section 5.6.2.1.
Signal number selection: Enter signal number to use for synchronization modes.

General Operations:

1) Accept new list
2) Discard changes to list
3) Create MIL file
4) Create module executable
5) View external interfaces

5.6.3. Defining an Application

Once a set of modules has been defined, an application is constructed by linking together the MIL files for each module interface, and adding the application definitions. The resulting file is a complete application MIL file, which is used by the bus program to start and run the application. The MIL files for each module are created by the templates (or programmers in other languages). These files are complete, except for the host names. Therefore linking these together means selecting and inserting the host names while concatenating the files together. The original files must be left untouched, so they can be regenerated when desired.

Declaring an application involves two types of declarations. The "tool" declaration instantiates a module as a node in the distributed application. The name of the node can be the name of the module which implements the node. It is also possible to instantiate multiple nodes using the same module. In that case, an explicit node name is placed in the tool declaration. Node names must be unique. Once a tool is declared, then an instantiation of an interface is defined by specifying node name-interface name pairs. Within a node, interface names also, of course, must be unique.

Finally, the interfaces are bound together by declaring mappings between input and output interfaces. At this point, the MIL file contains enough information to allow the bus program to start all the node programs and physically bind the interfaces together using whatever low-level transport structure the bus program implements. In current implementations that structure is always TCP/IP, and binding is done at the socket level. The actual process of passing input and output data back and forth is done by passing messages across the bindings, using the message formats that are defined in the interface declarations.

When the bus is run, two mode options are available, connection and keep alive. The default connect mode is to have all messages mediated by the bus program. Essentially all connections are made to the bus program, which acts as a router between the input and output half of every interface binding. Optionally, the direct connect mode allows connection directly between the node processes, which results in better performance. On the initial implementation, ControlCalc will use the standard mode. This is necessary in order to allow a single module to handle multiple interfaces to a spreadsheet template. Direct connect mode does not allow use of the mh_readselect option for monitoring multiple interfaces. In the general-purpose model we are implementing, a direct connect option would require a single interface per node, or multiple nodes to implement a set of interfaces to one spreadsheet template. The speed benefit of direct connection may justify the overhead of multiple nodes, and is a question we will address when we benchmark the system.

The second option, keep alive, overrides the default I/O access mode, which is to open and close the I/O channel involved on every message transaction. In most real-time systems, keep-alive would be required, due to the very high overhead of opening and closing I/O channels. However, most UNIX implementations impose a relatively low limit (15 - 32) on the number of open I/O channels per process. Again, this limitation argues for a single node per interface, since the nodes' non-Polylithic interfaces to the ControlCalc spreadsheet is not limited in this manner, being based on shared memory and signals, not I/O channels. An argument can also be made in favor of letting the keep-alive option be applied per-binding, rather the per bus, as is the case in the current Polylith implementation. In that way, non-ControlCalc languages that are best thought of as nodes with many interfaces each could accept the performance loss, while performance critical bindings could keep their channels open.

5.6.3.1. Creating the Application MIL file

Creating the application definition section of the MIL file is actually programming part of the application, since it contains statements which tie the module interfaces together, thereby instantiating

modules. However, this process is declarative programming, rather than sequential. It is therefore very amenable to a configuration utility front-end similar to the one described for the module interface definitions. Calling such a configuration utility within ControlCalc would also present a consistent interface style.

Programming and debugging issues also indicate the usefulness of a configuration utility. Using the construction technique for modules and their interfaces described in section 5.6.2.3., the result is a collection of MIL sections. In a complex system, this would mean a large number of small files, making it difficult to manage the application setup. An interactive configuration utility should therefore allow the user to display multiple sub-windows, each displaying a MIL module declaration section, to simplify browsing among the modules.

More importantly, both the application and the interface configuration utilities should provide interactive checking of syntax, preventing such obvious errors as:

1) Declaring duplicate node names
2) Using input interfaces in the output half of a bind declaration, and vice-versa
3) Using unknown names
4) Illegal syntax in general
5) Non-existence of hosts or executables

The current method of creating all the MIL file(s) is a text editor, followed by a compiler. Some typographical errors (such as naming mistakes) cannot be caught until run-time. An interactive utility would be able to provide immediate feedback, and browsing. Of course, errors involving unknown names should be posted as warnings, since it is not desirable to force, for example, definition of all interfaces before any bindings involving that interface. For a variety of reasons, we plan to have our application configuration utility maintain its own variation of a makefile which lists the modules involved without actually constructing a concatenated MIL file. However, all such files will be kept in text form, with a published syntax, so users who prefer a text editor, or have existing applications they want to add ControlCalc modules to, can do so without re-keying their application MIL section. Furthermore, the utility will be constructed as a stand-alone program, although a ControlCalc command option will be able to invoke it from within the spreadsheet.

The application configuration utility will provide the following operations:

1) Add/Delete modules (services)
2) Declare a host for a module
3) Add/Delete/Modify tools
4) Add/Delete/Modify bindings
5) Browse the networks host list (/etc/hosts)
6) Open a module interface section for editing (ControlCalc interfaces only)
7) Link/Compile/Write a MIL file on one or every host (if NFS is available)

5.6.4. Calling Interfaces from ControlCalc Applications

Up to this point, we have discussed only a mechanism for implementing message receipt and, optionally, reply to a message using the Polylith sink and function interfaces. It is equally important that ControlCalc programs be able to initiate message transfers, acting through Polylith source and client interfaces. Sink and function interfaces, as we saw, can be handled by treating the ControlCalc spreadsheet as a data base with computation on demand. Except for the computation itself, no programming beyond MIL-equivalent declarations are required. However, to implement source and client interfaces, interfaces must be invoked from within ControlCalc's functional programming

language. To do this, one additional configuration parameter is required: a declaration of which ControlCalc process implements the node.

5.6.4.1. Implementing Polylith Calls in ControlCalc

The basic approach we will take is to let the user access source and client interfaces as if they were functions in the spreadsheet, for the basic Polylith message passing functions mh_read and mh_write. The programmer would think of these interfaces as remote procedure calls, or as message passing calls. The function names would be the interface name. Because ControlCalc's language and compiler are, at this point, completely under the control of RTware, we can enhance the language to support dynamic function definitions such as this. Furthermore, ControlCalc's support for functional polymorphism relaxes the expected requirement of including the tool name as a parameter to the function. Without polymorphism this would always be necessary, since multiple instances of an interface name can exist, either because different modules use the same interface name or because a module is instantiated as a tool more than once. Access to the complete application and module MIL declarations allows the spreadsheet parser to see if the name is unique, and require an additional tool name parameter (which would have to be a constant string) only if the interface name is not unique within an application. Note that the compiler can determine from the MIL file how many parameters an interface declares. This means it is feasible, but perhaps not desirable for debugging, to distinguish between some common interface names by using differences in their parameter declarations and matching them to the function's parameter list. The only syntactic kludge required is that for client functions, output parameters would be required to be listed before input parameters (or vice-versa, by convention). Naturally, parameter ordering would have to match, since there is no parameter naming in the interface declarations.

Needless to say, it is also possible to provide direct implementation of Polylith's entire mh_ function set. Since ControlCalc supports text processing functions, it is reasonable to include all the select and query functions. ControlCalc's polymorphism inherently provides the varargs-style interface required for doing this.

5.6.4.3. Application Startup Method Conflicts Between Polylith and ControlCalc

As it currently stands, Polylith uses a static process model of programming which presents certain problems when being used with an interactive multi-processing system such as ControlCalc. The primary restriction is that each module must be implemented as a process which can be executed by the Polylith bus program. This is essentially how the entire application is started. Each module process must be a uniquely named executable and cannot receive application level command line arguments. Polylith does provide an attribute declaration statement for the MIL module section, which allows an instantiated module program to read tagged attributes. This is essentially a way of passing arguments to a process (or else process arguments are essentially a way of defining process attributes, your choice).

ControlCalc, on the other hand, operates with a small set (just three at this time) of executable modules which are configured dynamically according to declarations and functional programs. An executable ControlCalc program is actually compiled by the spreadsheet process, with code placed in shared memory sections. Being a multi-tasking spreadsheet, multiple code segments can be created, each potentially operating on any part of the entire three-dimensional spreadsheet data space. When the application is run, an executive program is invoked for each code segment, and is passed the parameters it needs to link to shared memory, link to various resources used by the code, locate the code segment, and execute the code segment when triggered.

The source/client implementation described in the above section requires that each ControlCalc program segment process be the binary for a node. To do this requires changes to either Polylith or ControlCalc's process model.

Changing Polylith to accommodate ControlCalc processes would involve making the bus dynamically configurable. Rather than requiring that all nodes be executed by the bus, the bus could start independently and allow processes to attach later. In attaching, a process would pass its declarations to the bus, the bus would validate the declarations against unattached services in its MIL declarations, and if a valid match were found, the bindings would be instantiated. At that point, the process would be free to make calls on the node's interfaces. This approach would allow ControlCalc to set up the necessary resource declarations for a process which is declared as the implementation of a node. Those resources would be equivalent to the MIL file declarations, and would be passed in to the bus.

Changing ControlCalc to accommodate Polylith involves generating copies of the ControlCalc general-purpose code segment master process, with unique names incorporating both the template key and the code segment key. Parameters required by these process and the names of the binaries would have to be added to the module's MIL section. Since ControlCalc creates the code segment and establishes the process parameters when the application is started, the launch of an application would involve the following sequence:

1) Start all ControlCalc templates involved in the application.
2) Each ControlCalc template process generates the necessary binaries and modifies the relevant MIL sections.
3) Rendezvous on all ControlCalcs completing their generation phase.
4) Compile the master MIL file.
5) Start the bus.

While this process seems a bit involved, it can be made completely transparent to the user. We are considering that a second bus be used to simply handle the startup process. That bus could be torn down when the application is running.

There is a simpler approach possible, but it will result in a significant performance loss. The other approach is to use the single node process that handles input interfaces to handle output interfaces on behalf of the ControlCalc program. This would mean that each Polylith call from ControlCalc would involve an additional message passing transaction to the node process. We consider this to be unacceptable for real-time systems.

5.6.5. Virtual FIFO Connections

ControlCalc currently provides a TCP/IP socket I/O option that allows spreadsheets to exchange data between "history cells". ControlCalc history cells are actually complete first-in, first-out (fifo) queues, with a user configurable maximum size. They can be used internally as rotating ring buffers that remember the last X values (hence the name history cell), or for buffering communication with other ControlCalc processes of I/O agents, without requiring operating system intervention. The socket I/O option transfers data from the output side, dequeuing it from the output history cell, to the input side, enqueuing onto the input history cell. A block transfer size configuration parameter gives the user control over the efficiency of this process.

Implementing an equivalent option with Polylith is quite straightforward with the system we are describing. Note that this virtualization of the history cell across two spreadsheets is inherently declarative, requiring no procedural programming by the spreadsheet user. Therefore it could be made an option in the Polylith module definition utility. An exact equivalence to the current implementation should be possible by using a Polylith array data type and the direct connect option.

5.6.6. Desired Performance Enhancements

There is one feature we will to add to Polylith to support the high performance networking usually required in distributed real-time applications. That is to make the "keep-alive" and "direct-connect" options configurable per interface. We think this could be done by establishing key-word object attributes which are used by the bus when the system is started. Object attribute statements like KEEP_ALIVE = interface_name or DIRECT_CONNECT = interface_name would instruct the bus to use these modes for bindings to the specified interfaces. They would also instruct ControlCalc to instantiate separate processes for interfaces using the direct-connect mode, or if more than the maximum number of interfaces allowed by the operating system are in the keep-alive mode. This approach would allow tuning the application to achieve a balance of performance and complexity.

6. Related Work

6.1. Spreadsheets

The ControlCalc spreadsheet is described by this author (Clarke, 1991) with full documentation available directly from RTware, Inc., Durham, NC.

In reviewing the literature on spreadsheets, there is almost a complete lack of academic consideration of the spreadsheet paradigm as a programming language. This is probably stems from the fact that computer spreadsheets were developed entirely by private business as a direct outgrowth of bookkeeping methods. Like word processing, spreadsheets are typically considered to be a specific application, thereby obscuring their many similarities to traditional programming languages and the wide applicability of the spreadsheet model. Commercially, however, many products allow users to present data in spreadsheet form, and some allow equation entry for data manipulation. There are, however, a number of discussions of specific applications developed with spreadsheets.

The DYNAGRAPH system by Aonuma (Kobe University, 1987) is an interactive simulation and modeling system based on an APL spreadsheet. A system for analytical chemistry is described by Freiser (1992). Financial modeling for decision analysis is described by Morrow (1991) and statistical analysis for social sciences by Bakeman (1992).

Commercial application of spreadsheet in control and data acquisition is found in the "Lotus Measure" system based on the Lotus 1-2-3 spreadsheet. That system allows data to be imported from industrial communicate links, to trigger recalculation and responses. However, the interpreted and single-tasking execution of the Lotus spreadsheet limits the data bandwidth and response times. The Lotus product is positioned as primarily a back-end processing tool for factory-floor data and not a control system.

DaDisp is a popular data acquisition and analysis product which use a spreadsheet format for presenting tables of real-time data. This system is strictly data acquisition, with no feedback capabilities, and uses the spreadsheet for data storage only, not expression evaluation.

Schelberg (1991) proposes a spreadsheet model as an alternative to the data-base model of distributed control systems. His proposal envisions the spreadsheet as an organization tool, using hierarchical 3x3 spreadsheet matrices for representing a distributed system.

6.2. Module Interconnection Frameworks and Distributed Applications

The Polylith system and underlying research is described by Purtilo et al (1991). The Polylith Users Manual is available from the University of Maryland.

Snodgrass (1989) and Purtilo and Snodgrass (1991) described module interconnection formalisms and the resulting Module Interface Languages and contrast their different systems, Purtilo's Polylith system and Snodgrass's Scorpion System (available at the University of Arizona).

Purtilo and Jalote (1989) describe the problems of distributed application prototyping and design and requirements for communication primitives and a bus organization.

6.3. Graphical Diagramming Systems

Binns and Vestal (1993) describe the MetaH structural diagram system for real-time architectures developed at the Honeywell Systems and Research Center. The Software Programmer's Manual is available from Honeywell or the authors.

Numerous commercial packages provide graphical diagramming systems, although not supporting distributed applications. Most prominent are LabView from National Instruments (Austin, TX), Labtech Notebook from Iconics (Foxboro, MA) and Virtual Engineering Environment from Hewlett-Packard.

6.4. Real-Time Distributed Systems

A major, ONR-funded effort addressing the design of real-time parallel software is the Proteus system, being developed by John Reif (Duke), Jan Prins (UNC-CH), Allen Goldberg (Kestrel) and colleagues as part of the Prototech project. The key approach to facilitating software prototyping and implementation is to provide a language for the very high-level expression of concurrency and real-time constraints, and techniques for their subsequent refinement to lower-level constructs and software platforms. The Proteus language is an architecture-independent parallel programming language which incorporates fundamental real-time primitives as well as novel high-level constructs for expressing time and resource constraints in a parallel setting. The high-level constructs attempt to fill a gap in the expression of resource requirements such as computational progress which conventional real-time constructs do not adequately address.

6.5. Schedulability and Performance Analysis

ONR-funded work has resulted in the PERTS system (Prototyping Environment for Real-Time Systems) being developed by Jane Liu and colleagues at the University of Illinois at Urbana-Champaign [LR+93]. PERTS supports the synthesis and validation of real-time systems by providing an integrated suite of system building blocks, performance models, and measurement and analysis tools which capitalize on recent theoretical advances for rigorously predicting real-time behavior. PERTS contains (1) techniques for the specification of flexible real-time programs, (2) an extensible library of scheduling algorithms and resource access protocols for time-critical applications, (3) tools for the performance prediction and validation of real-time systems built using different workload models and scheduling paradigms supported by these building blocks, and (4) simulation and measurement tools which extract the processing time and resource requirements from annotated source code. PERTS tools and building blocks are implemented in the C++ programming language and X-Window system, and can be customized and integrated into other prototyping and software engineering environments.

7. Potential Applications

This Phase II proposal's objectives together form a general-purpose distributed control programming system. Application areas for a distributed control system are very broad, including process control, remote vehicle control, machinery control and many others throughout the private sector and DoD. Within DoD, such applications can often be mission-critical and very high performance, where even fairly simple algorithms can be enhanced with distributed processing. In addition to these traditional control applications, the resulting system will be very useful for simulation, both in physical simulators like flight trainers and in computer hosted modeling.

All the following information in this section (section 7) is proprietary information.

7.1. Sample Control Applications

This section describes some typical applications of ControlCalc currently in use or development, and briefly outlines how distributed features would be advantageous.

7.1.1. Navelex: Naval Airfield Control Tower

In the first application, ControlCalc was used for a Navy airfield lighting system. The application involved controlling the runway light banks and monitoring relays to detect output faults. Monitoring was done remotely, using radio modem links to an optomux-compatible I/O network. The user interface was based on a graphical picture of the runways, with dynamic colors representing different states. Constraints were applied to operator actions through rules defined for different classes of lighting. Pop-up control panels are used to let the operator monitor and adjust lighting parameters. Dynamic color control is provided through tables of data in the spreadsheet, with a color mixing option for customization. This work was performed in about four months by a single engineer with limited software experience. It was successfully reviewed by the Navy and installed in Beaufort South Carolina in April, 1993. Not incidentally, the same project was attempted a few years before using third-generation programming techniques, but was canceled due to severe cost and time overruns.

After reviewing the system, Navelex has decided to take advantage of the multi-tasking nature of the system and consolidate a number of different control tower applications on one computer. They will be creating new applications using ControlCalc that acquire and display weather data, information from the tower security system, equipment monitoring sensors and data from the collision detection and avoidance computer. They plan to install display sub-systems at each controller's station, allowing each controller to use monitor and adjust conditions in any the these applications. In particular, the interface to the collision avoidance computer would be simplified with the MIF interface proposed for phase II. That computer will be a high performance UNIX system, communicating with TCP/IP. While it will be capable of operating standalone, processing radar data, they want to be able to extract data from the system and present it visually through ControlCalc. A MIF interface provides a simple means of doing this with both ControlCalc and the programs already implemented for the collision algorithms and radar data acquisition.

In addition to the Navalex contract, ControlCalc is being used by a British airport control systems manufacturer (OpalPort) for their next generation of control systems. Their applications typically involve distributed control of various airport sub-systems. Currently they use a proprietary token-passing network, but they are interested in upgrading that.

7.1.2. Orion Aircraft Radar Systems

RTware has also been working with the Naval Research Laboratories on a project which will be able to serve as an excellent demonstration site for the distributed control system resulting from Phase II. This project was recently awarded to RTware, with the selection of ControlCalc as the control language and GUI system. The project is the design of the next generation of radar control and operator interface for the Orion "sub-hunter" aircraft. RTware has discussed the possibility of using the project's laboratory as a test site for the proposed ONR MIF control system with the responsible engineer (Sharon Hrin) on the radar project. Her response has been very positive. Within her lab, there are a large number of heterogeneous systems which often must work together in distributed applications. With the radar project, for example, up to eight pods will be controlled simultaneously. Furthermore, the radar system will also be running in a laboratory simulation environment in which computers doing simulation and data analysis are interacting with the control system.

7.1.3. System Health and Performance Monitoring

In the last year, Encore Computer Corporation has produced a system health and performance monitoring and control system for their Infinity line of parallel processing UNIX computers. The application required porting ControlCalc to the 88K processor, to the UNIX operating system and designing and implementing an X-Windows control panel interface builder. About ninth months after first delivery of an alpha-testable port, the application is now being released as a standard, embedded feature of the Infinity operating system, under the trade name CommandCenter. Reviews by mainframe users and analysts indicate that the application provides state of the art visualization and control of the system's operating condition. It consists of 200 different graphic screens, about 2.0 MB of compiled ControlCalc code, 600 I/O points, extensive data logging and analysis, rule solving and operator actions. The system logs reports, sends mail, supports multiple logins, does continuous data acquisition and still takes less than one percent of system processing time. The entire application was developed by a single project engineer, with much of this time spent qualifying the porting and development work required. He estimates that the actual application development work, including I/O drivers, took about six months. This application is relevant to DoD because Encore has a number of contracts with DoD for mainframe replacements, including a system at the Naval Research Laboratory.

The latest phase of this work was to make the system fully distributed using Encore's Reflective Memory System (RMS). RMS is a high-speed direct memory to memory link between system nodes in different backplanes. For this technology, RTware provided extensions to the shared memory I/O feature which is already available on the UNIX version of ControlCalc. The extensions allowed the application to map reflective memory regions into ControlCalc shared memory blocks, which can then be accessed using ControlCalc's standard I/O functions. The result is that each node of a cluster UNIX installation is running a local ControlCalc application which is responsible for data acquisition, analysis and response. Consolidated and exception data is posted to each node's reflective memory port, where the master control and operations application can access it.

The reflective memory approach has large advantages in performance over TCP/IP, and works well for continuous monitoring and control. However, it would be very useful to also have a message passing option that did not require low-level implementation through shared memory and semaphores. The MIF approach will provide just that. Furthermore, the success Encore has had with CommandCenter has interested them in producing a general-purpose version of the system that would be compatible with the standard system 3.2 UNIX kernel interface. The product would then be compatible with all the major UNIX platforms, including those from HP, NCR, Sun and DEC.

The primary technical requirement for a general-purpose computer cluster CommandCenter is a message passing I/O option, essentially the MIF concept. Such UNIX clusters do not usually have tightly coupled shared memory like Encore's RMS, but always have the standard TCP/IP networking connections. Because ControlCalc is completely programmable, the general-purpose CommandCenter application can be customized to take advantage of any features, like RMS, that a particularly architecture provides.

7.1.4. Turbomachinery Control Systems

ControlCalc is already being used for control of turbomachinery, of the type typically used on large Naval vessels. A major turbomachinery controls manufacturer has now released their new generation low-cost control product, which is a ControlCalc application. After starting work on March 1, 1993 they had the system in test on May 1, 1993. No significant problems were found in alpha or beta testing, so they are now starting to market it. The general manager of the company informed RTware that, if anything, we undersold the productivity improvements from ControlCalc. Their application consists of a full authority, multi-loop controller with a complete graphical user interface for monitoring and adjusting system parameters.

It is interesting to note that the turbomachinery company's software engineering staff did not believe that a spreadsheet programming paradigm would be able to do the job at all, let alone in three months. In this case, one control engineer simply did it. This resistance to a much simpler programming method is typical of software engineers, as we saw in some discussions at the Prototech conference. It is however, not justified. We have found that non-programmers can produce highly complex, reliable control systems even without the benefit of modular programming techniques. This is the experience of spreadsheet users in general, which is why spreadsheets are one of the top three applications in use today.

ControlCalc has also been selected as the control system for all the turbomachinery in western Canada's natural gas pipelines. After a two year pilot project competition with GE-FANUC (a world leader in programmable controllers) at WestCoast Energy Corporation, ControlCalc was found to be superior to the programmable controller in performance, flexibility, ease of use and operator interface. The competition involved two separate installations running a pumping motor unit in production. Deployment to over 200 installations is expected to begin in 1994.

Finally, a Naval subcontractor, Hyde Marine, has specified ControlCalc as the turbine control system in a proposal for a Navy surface ship control system.

In all these turbomachinery applications, distributed control is an important issue. A turbine is always a component in a larger system, whether that is a ship, a pumping station or an oil refinery. At this point, ControlCalc includes a standard industrial network option called Modbus which is used to tie together PLC's in such installations as oil refineries. However, Modbus is limited to moving blocks of data around, and does not provide any type of high-level function call interface. It also has very limited data types (16-bit integers only, for example) and has a fixed, master-slave configuration. It requires specific knowledge about the layout of data in each slave by the master, and so does not support modularity very well. Within, for example, a ship-board application, there may be multiple turbines each with their own controllers, hooked to supervisory systems and ship-wide communication networks. Constraint calculations such as power demand and damage control would be derived from a variety of data sources, not just the turbines, and processed through various computer systems. The turbine control systems would have to be hooked into the ship-wide control networks involving various types of computers and software. A very flexible LAN topology is required to maintain high mission availability in the face of hardware damage and rapidly changing network traffic patterns. A consistent interconnection system is required to field such integrated systems in a timely and reliable manner.

7.1.5. Simulation and Prototyping

One application area that has not been explored but has great potential is simulation and prototyping. One ControlCalc customer is doing simulation of a artificial diamond production process, and reports good results. RTware has also developed a simple temperature response model in spreadsheet form which is used to demonstrate feedback response to process control algorithms. In general, spreadsheets are used extensively in financial modeling and simulations. In Japan, Aonuma (1987) has developed an

interactive modeling system named DYNAGRAPH using an APL spreadsheet. RTware knows of a number of control engineers who model their algorithms in commercial spreadsheets before implementing them in control systems.

The spreadsheet paradigm is useful for simulation because simulation usually involves large arrays of data and sets of modeling functions (such as heat transfer functions) that operate on these arrays. The ease of handling data tables in the spreadsheet, and the immediate presentation of results to the user in tabular form are key features of a spreadsheet. When you add the ability for a spreadsheet to evaluate continuously, do multi-tasking, read and write direct I/O devices, drive on-line graphics and run at full compiled speeds, real-time simulators become feasible using the same paradigm. Potential applications include training simulators, combat arena simulation, war-game modeling and some of the real-time components of virtual-reality systems.

RTware has been told that a proposal for naval battle-arena simulation has been submitted to the Malaysian Navy by the Malaysian subsidiary of Encore, and that response has been favorable.

Prototyping is another area where rapid, interactive development techniques are required. The spreadsheet paradigm has a natural ability to present results to the user without any user interface coding being required. This makes it easy to program, test and interactively modify an application. With ControlCalc's multi-tasking and real-time scheduling capabilities, complex systems can be rapidly prototyped. The incremental nature of a spreadsheet is also a help. As long as a set of spreadsheet formulas in internally consistent, it can be evaluated and the resulted observed. Tables of data loaded from files, for example, can provide data sets for prototype function blocks to operate on. When implemented within a MIF, such prototype applications can be presented as modules, without the overall applications knowing how they were implemented. The sufficiency and efficiency of the specified module interfaces can be immediately tested by simply entering data into a spreadsheet and binding the interfaces to those tables of data. Simple counters and messages can be used to simulate the result of the intended algorithms, allowing high level testing to proceed without requiring full low level implementation.

8. Relationship with Future Research and Development

All information in this section is highly confidential and completely proprietary to RTware Inc.

It is expected that this proposed Phase II effort will complete the R&D phase and make possible the implementation of the applications described above. Phase III funding will not be required from DoD, as sufficient private and DoD application opportunities exist to make the resulting product commercially viable. The remainder of this section presents a brief business plan for the commercialization of the products developed by the phase II effort. At the end of this section, three letters are presented from organizations with ongoing relationships with RTware. These letters describe specific opportunities for the distributed ControlCalc/MIF system.

8.1. Competitive Advantages

The product that results from this Phase II effort will have very significant competitive advantages in the control systems market place, as listed below.

8.1.1. Modularity

MIF capabilities reduce system design cost in the following ways:

1) Software developers to work in parallel with a reduced chance of side-effect bugs
2) Software components can be reused
3) Language independence
4) Hardware independence
5) Reduced effort to enhance existing software systems
6) Simultaneous use of top-down and bottom-up design

In additions to these well understood benefits of modularity, the ControlCalc/Polylith implementation has some unique advantages. The modular nature of Polylith itself makes it easy to change the communications mechanism and port the system to different platforms while maintaining a consistent interconnection methodology and syntax. The ability to declare the module interconnections in spreadsheet form means that interconnection design can be immediately exercised manually and through spreadsheet what-if simulations. The modular design can also be carried down into a MIF node using the modular function-block technique. Low-level I/O and function programming in the spreadsheet allows specific modules to be rapidly constructed, reducing the chances that problems discovered in implementation will impact the high-level design. Finally, the ability to call software functions through a standard, compiled sub-routine linkage means it is easy to insert existing software into the same template used for doing the prototyping and design.

8.1.2. Performance

Compiled execution of control functions entered in spreadsheet form has been proven to out-perform all PLC's, and to be comparable to compiled languages such as 'C' and Ada. This means that the power of a spreadsheet language for prototyping and simulation can be applied to the run-time implementation. While performance is a simple concept, ControlCalc's ability to achieve third-generation language performance from a fourth-generation language is a primary factor in its current success.

8.1.3. Ease of Use

A spreadsheet is well understood to be easy to use by people from a wide range of disciplines, not just computer science. In fact, most technical disciplines work within a paradigm of rules and mathematical

formulations that fit much more naturally into a spreadsheet format than into the recursive, sequential paradigm of third-generation programming languages. As AI research has shown, most technical knowledge is contained in sets of rules, and a spreadsheet is a simple way of entering and evaluating sets of logical and numeric rules.

The strong emphasis on graphics in the ControlCalc/Polylith MIF system is an important component of ease of use. In large systems, modularity must be visualized through graphical interconnection diagrams. And of course the final applications must have graphical user interfaces. The MIF system will have both the modular diagram interface and a uniquely powerful mapping approach to driving the output of standard graphical user interface editors.

8.1.4. Integrated Environment

A consistent software environment for application design yields competitive benefits in both the development process and the actual design of an applications. The system allows decision on the topology of a distributed system to be made without artificial constraints imposed by the different capabilities of the software components involved.

8.2. Current Major Market Areas and Opportunities

ControlCalc is currently successful in a broad range of market areas, many of which involve distributed control and have direct application to DoD systems. A summary of actual control applications currently done with ControlCalc follows:

Machine Control: Turbines, commercial industrial machinery, transit vehicles, injection molding machines
Data Center Monitoring and Control: CommandCenter applied to the general market
Factory Automation: Automobile Parts Plant, Steel Mill, Aluminum Mill
Automated Test Systems: Engine Test Cells, Large Vehicle Wind Tunnels, Rocket Testing, Destructive military equipment testing
Laboratory Instrumentation: USFDA X-Ray testing and qualification program
Airfield Systems: Control tower automation, lighting systems, loading ramp controllers, integrated airfield systems
Process Control: Chemical processing and emergency alarm management and response

8.3. Future R&D with commercial partners

There are a number of areas in which R&D will continue, building on the results of this phase II effort. These areas include particularly enhancement of the graphics and extending the system to other platforms as they become established. RTware believes that the result of the phase II effort will be products which will generate sufficient interest to fund continued development and enhancement through internal funding and joint contracts with RTware's partners, end-users and distributors. To justify this belief, letters from some of RTware's larger customers/partners are attached at this point. In general, these letters cannot commit the organizations involved to specific funding, but do demonstrate serious and sustained interest in purchasing and/or distributing the products that will result from the phase II effort, based on their existing use of RTware's ControlCalc system. Details on these partners follow:

Encore Computer Corporation is a supplier of high-performance, parallel processing UNIX systems to the data center market. Their ControlCalc-based application, CommandCenter, is currently shipping on all Encore Infinity machines. Encore has funded RTware's port of ControlCalc to UNIX and X-Windows over the last year and a half. Total funding has exceeded $ 150,000 and is expected to increase somewhat over the next two to three years. RTware has a renewable five-year contract with Encore which provides software royalties for every Infinity system shipped. Royalties from the Encore product

alone are now over $30,000 per quarter and are projected to exceed $500,000 per year within two years. Encore is preparing plans to port the software to the general-purpose UNIX market, but has limited funds to do this. This SBIR proposal's technical goals mesh exactly with the Encore's requirement for the next phase of the work. Funding for phase II will dramatically improve the chances of the project succeeding.

Motorola Computer Group provides VME boards and systems to a broad range of markets, including real-time control, automation and data processing, with a significant presence in DoD markets. Motorola is now offering an integrated package of hardware and software using RTware's ControlCalc system. Funds for research and development under this project are very limited since the major funding is directed at worldwide sales and marketing through Motorola's electronics distributor, Arrow Electronics. As indicated in the attached letter, Motorola has hardware and operating systems platforms which can take full a antage of the results of this Phase II proposal. In fact, high-level corporate commitment to the curren. project was difficult to achieve since that systems could not be used with the current, non-distributed version of ControlCalc.

Gespac S.A. is a Swiss manufacturer of computers specifically designed for low-cost real-time control and automation applications. Gespac also has a real-time GUI software package which RTware has integrated into the ControlCalc system. That integration allowed the complete elimination of the C programming requirements of Gespac's original package, and demonstrated the effectiveness of driving GUI systems by mapping them into spreadsheet functions and commands. Gespac distributes and supports ControlCalc worldwide, with applications in place in the U.S., Europe and Japan. Gespac is heavily involved in the industrial networking market, especially with the emerging FieldBus standards. They have a distributed Fieldbus system with intelligent local nodes capable of running ControlCalc compiled code. Marketing executives at Gespac indicate that the DCS-II concept hosted on Fieldbus has very exciting market potential. Gespac does not, however, have the resources to fund the fundamental research and development efforts proposed here. They are capable, however, of porting the resulting system to their hardware once the building blocks are in place.

8.3. Business Potential of the Product

At a time when only one major U.S. PLC manufacturer (Allen-Bradley) remains an American company, continued success of U.S. innovation in the controls market is vital to long-term U.S. national interests, and to maintaining the lead of the U.S. military in advanced software technology. Market research figures in the trade press put the distributed controls market at over fifteen billion dollars annually, worldwide. An increasing portion of that amount (now over 30%) is devoted to software and integration services. The market is dominated by the large controls suppliers such as Westinghouse, Honeywell and Fisher, but has many small to medium sized sectors. In chemical process control, one small software company has annual sales of over 10 million dollars based on distributed monitoring applications alone. One software company that sells just the operating system platform (VxWorks) used in a many Naval applications also has annual sales of over ten million dollars, with strong continuing growth. The market itself is expanding rapidly despite worldwide recessionary conditions. Success of the distributed ControlCalc/Polylith product in even a few limited sectors can be expected to create a company of at least that size. General adoption of the ControlCalc distributed spreadsheet paradigm by the controls industry would have an impact comparable to the adoption of the electronic spreadsheet by the financial industry.

8.4. Letters of Support

The following pages provide copies of letters of support for this phase II proposal from various organizations. The originals of these letters are on file at RTware and may be inspected at any time.

BOOZ·ALLEN & HAMILTON INC.

4975 LACROSS ROAD, SUITE 313 • NORTH CHARLESTON, SOUTH CAROLINA 29418 • TELEPHONE: (803) 529-4900 • FAX: (803) 566-0801

12 October 1993

To: Office of Naval Research

From: Mike Shuler

Subject: ControlCalc

Under current contracts/delivery orders from NAVELEXCEN Charleston, SC, Booz• Allen & Hamilton, Inc. (BAH) has developed a control system based upon RTware's ControlCalc real-time spreadsheet for use in all Navy and Marine Corps Air Traffic Control (ATC) towers. In addition, NAVELEXCEN Charleston is planning to develop a VME-based system for the ATC towers to consolidate numerous data gathering, processing, communications and display functions into an integrated system involving multiple displays, application programs, operating systems, etc.

It has been brought to my attention that RTware, Inc. is proposing a development project for the Office of Naval Research under phase II of the SBIR program. According to the material I have reviewed, this proposal encompasses the development of a distributed control system based on the ControlCalc real-time spreadsheet and a high-level interconnection framework. Given the current and anticipated NAVELEXCEN projects involving ControlCalc mentioned above, I would like to express our support for this project.

The system developed by BAH is dubbed the Airfield Lighting Control System (AFLCS), although it is not limited to airfield lighting control. The AFLCS consists of numerous distributed radio-controlled remote I/O devices which communicate with a host PC in the ATC tower. A semi-scale graphical display of the airfield and all controllable functions is presented on a 17-inch SVGA monitor in the tower. Using a trackball or mouse, the air traffic controller simply clicks on a particular runway, taxiway, or other graphically represented function depicted on the screen to initiate a control action. Special windows can be pulled up as troubleshooting aids to analyze system operation from the tower.

The remote devices consist of digital and analog I/O modules by Phoenix Contact, Inc. Communication between the ATC tower and the remote devices is implemented using the industry-standard Optomux protocol via RF modems configured in a point/multi-point half-duplex arrangement. The system provides continuous status updates to the tower display and immediate execution of operator-initiated commands, along with

communication timeout and recovery per-controller. Communication loss with one or more I/O controllers does not prevent communication with the remaining controllers.

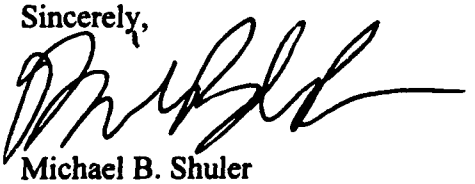
Although the airfield lighting and other control functions are similar at the various airfields, the number of I/O points will change from site to site depending on the airfield configuration. In addition, implementation of a site-specific control functions and/or control interlocks require customization of parts of the control spreadsheet. However, software configuration control among the 40-plus airfields is made relatively simple due to the high-level, 4GL nature of ControlCalc and the G-Windows window manager by Gespac.

Reviews by operational and technical personnel have been very positive. One of the major strengths of the system is its real-time multi-tasking capabilities. When implemented on the planned VMEbus, the system will be expanded far beyond its current airfield lighting control functions, and will require multiple application programs (existing and new) and operating systems to be implemented concurrently. In addition, network communications will be required between this system and other systems such as RS-6000 Unix-based radar processors.

The distributed control architecture proposed by RTware could possibly be used to our advantage in our VME-based system. It would then be possible to import/export data and procedural interfaces from system to system within a consistent framework (the Module Interconnection Framework) even when the systems are programmed in different languages and run on different types of processors. This would simplify and rationalize the interconnection between custom-written Ada or Fortran programs on the RS-6000 (radar processing, flight progress, traffic analysis etc.) and spreadsheet logic with graphical interfaces built with the ControlCalc system. Other serial I/O subsystems such as weather data collection could be processed by ControlCalc on the VMEbus, presented to the control tower personnel and simultaneously routed to other dedicated computer programs and data logging systems. There are clearly advantages to accomplishing the above via a standardized interconnection framework instead of custom-built interfaces.

Although NAVELEXCEN Charleston will continue to support and enhance the PC-based standalone AFLCS, its functions will eventually be incorporated into the VME-based solution. While NAVELEXCEN Charleston has not been funded to develop such a general-purpose framework as proposed by RTware, its existence would provide alternatives not presently available for implementing the VME-based product described above.

Sincerely,



Michael B. Shuler
Associate

MOTOROLA
Computer Group

October 8, 1993

To: Dr. Ralph Wachter, Office of Naval Research

From: Jim Albers, Senior Staff Engineer, Motorola Computer Group

Re: RTware and ControlCalc

I would like to describe Motorola Computer Group's involvement with ControlCalc and add my enthusiastic support for RTware's phase II SBIR proposal to the Office of Naval Research. The proposed use of a Module Interconnect Framework (MIF) system would really increase the types of applications and the acceptance of RTware's unique spreadsheet-based programming system. Motorola Computer Group (MCG) cannot, of course, formally commit at this stage to commercial use of any product that may result from that work. However, our work with ControlCalc and the interest from our customers makes it very likely that MCG will do so.

MCG already offers ControlCalc as part of an integrated control system kit, the 162Kit, This system is based on the MVME162, a single-board computer using the Motorola 68040 processor. The board offers:

- 4 IndustryPack (IP) slots that host a wide variety of graphics, communications, and industrial I/O modules,
- optional SCSI and Ethernet
- 68040 or 68LC040
- up to 4MB of DRAM and 2MB of battery backed SRAM in various configurations
- optional VME interface with high-speed D64 block transfer.

Running any of the leading real-time kernels and operating systems, the MVME162 platform is especially suited for high-speed control, data acquisition, and communications in a wide range of environments. With a good real-time OS, this hardware platform is well suited for custom programming efforts. Before our MVME162 can be widely adopted by control engineers (non-programmers) as a real-time control applications platform, we require good real-time application enabler packages. ControlCalc meets this need very well.

We find that the extremely high performance of ControlCalc's multi-tasking, compiled spreadsheet does allow the 162Kti to handle applications that previously required dedicated Programmable Logic Controllers (PLCs) or costly custom programming efforts.

For example, veteran control system engineers that we work with have discovered that ControlCalc is an ideal tool for building and maintaining control systems characterized best as complex state machines (e.g. oil and gas plant shutdown systems). The complex state machine specifications (with state transition matrices as large as 400x400) translate directly into ControlCalc spreadsheets. Changes can be made and verified reliably and rapidly compared with existing control systems.

5620 Smetana Drive, Suite 100
Minnetonka, MN 55343-9611
Telephone: (612) 932-1505 - FAX: (612) 932-9888
Email: albers@chg.mcd.mot.com

Motorola is very interested in a distributed version of ControlCalc running with X-Windows and on multiple platforms, and for a very practical reason. While the current commercially available version of ControlCalc does run on Motorola VME systems under the OS-9 single processor OS, Motorola would prefer that the product run under our own VMEexec operating system. VMEexec fits very will in the model proposed by RTware for the SBIR program. VMEexec is a multi-processor system, where program development is done on a CPU running standard System V Release 4 UNIX, and the user interface runs on any X server. Real-time processing is done with closely (or loosely) coupled CPUs in the same backplane or over a TCP/IP connections. VMEexec is a full set of software development tools, inter-processor communication mechanisms, drivers and libraries built around the pSOS+ kernel. The VMEexec system today has full use of C, FORTRAN, C++ (GNU or USL Cfront 3.0.1), X11R5 libraries, OSI, TCP/IP and X.25 protocol stacks, and so on. Motif clients provide high-level source debug of running target code in C, C++, and FORTRAN.

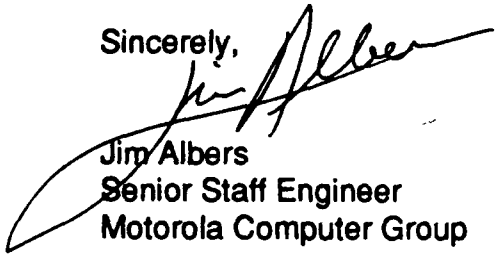
This architecture conforms exactly to the DCS-II concept proposed by RTware. Since multi-processor operations over the backplane and over TCP/IP differ only in latencies (not in protocols), the VMEexec platform can support both forms of distributed processing proposed by RTware.

We are also very aware of the benefits provided by the sort of interconnection framework being studied by RTware. In many applications, real-time subsystems must communicate with each other and with other back-end processors. Use of a standard interconnection scheme that hides implementation details of subsystems (or modules) is an obvious requirement. Previous work on distributed operating systems at Motorola (the Computer X CXOS project) demonstrated the technical and market demand for a system built on a truly distributed message passing microkernel. The primary advantages of CXOS and the DeltaRT systems based on CXOS were also its downfall. By providing highly integrated distributed graphics, DBMS, and application enabler tools (the Rapid Development Platform), the DeltaRT platform enabled the solution of many tough distributed processing problems. However, the proprietary CXOS core of DeltaRT limited its adoption.

The need today is to host distributed systems of open platforms, especially UNIX.

Again, Motorola can make no commitment at this point. However, the proposed development efforts fit very will with both our immediate work with RTware, our medium term plans for VMEexec, and the long term need for the adoption of a standard interconnection framework in the real-time market.

Sincerely,



Jim Albers
Senior Staff Engineer
Motorola Computer Group

Encore Computer Corporation
6901 West Sunrise Boulevard
Fort Lauderdale, Florida U.S.A.
33313-4499
Telephone (305) 587-2900

October 8, 1993

Office of Naval Research
Dr. Ralph Wachter

Dear Dr. Wachter:

For the last year, Encore Computer Corporation has been involved in a major software development effort using RTware's ControlCalc software product. The result of this effort has been our CommandCenter system software package. CommandCenter is a sophisticated control and monitoring system for Encore's Infinity series of massively parallel UNIX machines. Infinity systems are positioned as "right-sized" mainframe replacements, and currently run the highest Transactions per Second benchmarks ever recorded on the Oracle relational database. Infinity machines have been shipping with CommandCenter, over this year to such major accounts as The Naval Research Laboratories, DISA Megacenter, and has been approved for the DMRD924 program of the Department of Defense, and to major commercial accounts such as EDS and the IRS.

In all applications, CommandCenter has been acclaimed as a state-of-the-art system for on-line system monitoring and control. Encore was able to produce a highly graphical interface and a consistent, easy to understand presentation of numerous system parameters using ControlCalc as the core of the product. CommandCenter uses a data agent to read and write data from the operating system kernel, placing all data in the multi-tasking ControlCalc spreadsheet. Our entire system of over 300 screens, dozens of tasks and continuous I/O was created without a single line of traditional code, and in less than one person-year of development. Due to the compiled, multi-tasking capabilities of ControlCalc, the CommandCenter application runs very efficiently, typically using less than one percent of cpu time while doing continuous system monitoring.

ControlCalc was selected for this application after an extensive review of the many products available in the UNIX market. ControlCalc's combination of high performance, extendibility and programmability was unique. Our experience with the system has proven that the spreadsheet programming approach to on-line systems is extremely productive, and compares very favorably to application-specific packages and to third and fourth generation programming languages. Encore has provided ongoing funding to RTware, Inc., the owner of ControlCalc, to port the system to the Infinity and support our development efforts.

Our current work on CommandCenter focuses on Encore's exclusive Reflective Memory System (RMS). RMS provides a high-speed, memory to memory link between the multi-processing nodes of large scale data center installations. RMS performance and the associated software in the Infinity File System results in Infinity's unprecedented

ENCORE
COMPUTER CORPORATION

scalability in even large, transaction-oriented mainframe applications. CommandCenter will soon be released with RMS support, allowing simultaneous monitoring of remote nodes without any network traffic overhead. Each node in the system runs a local copy of CommandCenter and communicates only its results over RMS. The result is minimal traffic between nodes and robust stand-alone operation in event of a node or communication failure.

Looking ahead, Encore expects to continue developing CommandCenter and working with RTware to enhance ControlCalc. One area of particular interest is extending the CommandCenter system to handle heterogeneous, large-scale systems using network communications such as FDDI or Ethernet. For this reason, we are very interested in the proposal RTware is submitting to ONR under the SBIR program. We have reviewed the technical proposal and find that it fits in very well with our long-term plans. Encapsulating ControlCalc within a Module Interconnection Framework (MIF) will allow us to present a product which is portable, extensible and can interact with existing systems.

While Encore Computer Corporation will continue to fund development of RTware, we are aware that major enhancements such as MIF and porting a wide range of platforms will require support such as the SBIR program is specifically designed to provide. We can assure ONR that the resulting product will be commercially viable and that Encore specifically will be interested in using the product.

Sincerely,

Ziya A. Aral

Ziya Aral
Vice President, Engineering &
Chief Technical Officer

ZA/kb

SOUTHWEST RESEARCH INSTITUTE

6220 CULEBRA ROAD • POST OFFICE DRAWER 28510 • SAN ANTONIO, TEXAS USA 78228-0510 • (210) 684-5111 • TELEX 244846

ENGINE, FUEL, AND VEHICLE RESEARCH DIVISION
TELECOPIER: (210) 522-5720

October 18, 1993

Richard Clarke
RTWare, Inc.
714 9th St., Suite 206
Durham, NC 27705

FAX: (919) 286-6629

Dear Richard:

As we have discussed over the last year or so, SwRI is very interested in an Iconic programming interface, as in National Instruments' LabView for Windows (c) or MatrixX. The ability to provide a polymorphic data flow, while not the driving factor, certainly allows our users the opportunity to extremely quickly flesh out their ideas and turn them into working programs.

Additionally, as our high speed combustion needs continue to grow, it would be helpful if there were a way to distribute the processing programs across multiple CPUs. It is very expensive to purchase high-speed super micros to analyze data for all of our 40-50 test cells. If ControlCalc supported a distributed mechanism, this concept would be easier to implement and nearly transparent to the user.

Also, as we have discussed, our need for this type of interface has finally made management willing to invest the necessary resources, such as our manpower or division dollars, on the development of such a product. It would be helpful/useful for us to have this, and as such, we are willing to cooperate with RTWare in the development of this product.

Please let me know of future plans for this style of interface and the possibility of distributed processing. We enjoy our close relationship with RTWare and foresee these products figuring heavily in our test/controls development future.

Sincerely,

Josef Zeevi
Research Engineer
Department of Vehicle Systems Research
Telephone: (210) 522-5389

JZ:jw

SAN ANTONIO, TEXAS
HOUSTON, TEXAS • DETROIT, MICHIGAN • WASHINGTON, DC

Gespac, Inc

50 W. Hoover Ave.
Mesa, AZ 85210
Tel: (602) 962-5559
FAX (602) 962-5760

TO: Office of Naval Research

FROM: Fred Strouse, General Manager, Gespac, Inc.

DATE: October 18, 1993

SUBJECT: ControlCalc

I would like to express Gespac's strong support for the ControlCalc/MIF Phase II SBIR proposal submitted by RTware to the Office of Naval Reseach. Our long-term involvement with ControlCalc shows that the spreadsheet idea is very powerful, and very practical. Enhancements to support distributed control systems are definitely the important next step for this product.

Gespac Inc., and our parent company, Gespac S.A., promote and distribute ControlCalc worldwide for demanding real-time applications. ControlCalc has been combined with Gespac's real-time graphical windowing environment (G-Windows) and graphics user editor (G-View) to provide a very simple method of driving custom graphical interfaces without any C programming required. Applications in both industry and government have shown that ControlCalc completely lives up to its promise of hard real-time performance in a spreadsheet environment.

Gespac supplies intelligent distributed I/O systems using various technologies, including Filbus, FIP, and BITBUS .We are extremely interested in the RTware proposal's goal of allowing multitasking spreadsheet applications to be distributed across tightly or loosely coupled networks. Distributed real-time control is a major target market area for Gespac. Distributed ControlCalc will be very important to us as we penetrate the DCS market.

While Gespac cannot fund an effort of the scope outlined in RTware's proposal, I can assure you that Gespac will participate in commercialization of the resulting technology. Gespac already is an OEM for RTware, and we expect that relationship will continue to expand.

As part of our cooperation with Rtware, Gespac supplies to Rtware, for example, all the equipment necessary for Rtware to support our line of rugged, real-time computer systems and I/O.

Again, I would like to say that Gespac fully supports this proposal and looks forward to exciting marketing opportunities for the products that will result.

Sincerely,

Fred Strouse,
General Manager, Gespac Inc.

9. Key Personnel

1) Richard J. Clarke, Principle Investigator and President of RTware, Inc.

Mr. Clarke graduated with highest honors with a B.A. in Sociology from U.C. Berkeley in 1977. In 1979, Mr. Clarke entered Rensselaer Polytechnic Institutes' graduate school (Hartford Graduate Center). Mr. Clarke maintained a 4.0 GPA and received a Masters in Computer Science in 1981. After graduation, Mr. Clarke continued to work in industry, specializing in real-time control and factory automation systems. Prior to founding RTware in 1990, Mr. Clarke held Principal Engineering positions in Data General's real-time and UNIX operating system kernel groups, was supervisory of control systems for a large industrial machinery manufacturer and was software manager for real-time operating systems at a VME systems company. Mr. Clarke has published one paper describing the application of spreadsheets to real-time control (Clarke, 1991).

2) Anthony Kostichka, Real-Time Software Engineer, RTware Inc.

Mr. Kostichka received his B.S. in Electrical Engineering from the University of Wisconsin in 1990. Prior to joining RTware in 1993, Mr. Kostichka worked for three years in the UW chemical engineering laboratory designing data acquisition and control software and instrumentation. Mr. Kostichka also worked extensively with Graphical User Interfaces and has expertise in object-oriented graphic, C++ programming, MS-Windows and Macintosh programming. At RTware, Mr. Kostichka is responsible for I/O drivers, including interfaces to industrial networks, and for sections of the graphical interface of ControlCalc.

3) Steven Burnett, Technical Writer, RTware Inc.

Mr. Burnett received his M.S. in Technical Communication from North Carolina State University in 1991. Prior to joining RTware in 1992, Mr. Burnett was responsible for technical reports to the Environmental Protection Agency from a EPA contractor. At RTware, Mr. Burnett is responsible for the ControlCalc documentation set, including User's Guides, Reference Manuals, Tutorials and Installation Guides.

RTware will also be hiring an additional senior software engineer and one junior software engineer for the phase II effort. The senior engineer will be at or above the Masters of Computer Science level, with expertise in communications, multi-processing, language design and graphics programming. The junior engineer will have a B.S. in computer science, and will provide programming and operations support as required by different stages of the effort. RTware will also contract with a senior, PhD-level individual with expertise in real-time languages, distributed processing and visualization for fundamental design work. This individual is listed in the cost proposal as Principal Engineer.

10. Facilities and Equipment

RTware's Durham N.C. offices will be the site of at least 90% of all work done in this project.

A number of computer systems will be required to carry out the Phase II effort. These include standard workstation systems for software development and specialized real-time systems for testing and porting the software to target environments. A detailed equipment list is presented in the cost proposal. Note that exact models and prices are not in the cost proposal, as this is subject to change, within the budget figures presented.

11. Subcontracts or Consultants

Consulting services will be required both to minimize the time to learn and integrate third-party software and to make any modifications required in a timely manner. Consulting will be obtained from Dr. James Purtilo at the University of Maryland for the Polylith system, and from Dr. Jane Liu at the University of Illinois for the PERTS system. In both cases, consulting may be done directly by the professor, or by qualified graduate students or researchers, depending on the nature of the work required. In all cases, hourly rates and total costs will not exceed the budget figures presented in the cost proposal.

Specific tasks expected to be performed by consultants include:

1) Enhancing Polylith to support per-instance declaration of keep-alive mode.

2) Enhancing Polylith to support dynamic registration of module interfaces.

3) Porting Polylith to communication protocols other than TCP/IP.

4) Constructing an interface to the PERTS system to provide on-line access from ControlCalc.

5) Reviewing the real-time scheduling and synchronization primitives for completeness and efficiency.

12. Current Pending Support

None.

There is no current pending support for this proposal by any Federal agency, DoD component or the ONR.

13. Cost Proposal

13.1. Name of offeror: RTware Inc.

13.2. Home office address:

RTware, Inc.
714 Ninth Street, Suite 206
Durham, North Carolina 27705

13.3. Location work will be performed:

714 Ninth Street, Suite 206
Durham, North Carolina 27705

13.4. Title of proposed effort:

Module Interconnection Framework for Software Producibility

13.5. Topic number and topic title from DoD Solicitation Brochure:

N92-112, Module Interconnection Frame for Software Producibility

13.6. Total dollar amount of the proposal: $ 730,227

13.7. Direct material costs:

Item #	Qty.	Description	Unit Cost	Total Cost
1	3	Sparc Classic Workstations with Professional C development Software, color monitor and disk.	$ 5,000	$ 15,000
2	1	LynxOS real-time operating system, including 2 years support and professional OEM development licenses.	$ 12,000	$ 12,000
3	1	DSP operating system and software development package.	$ 4,000	$ 4,000
			Total Direct Material Cost:	**$ 31,000**

13.8. Material Overhead Rate: N/A

13.9. Direct Labor:

Name	Responsibility	Hourly Rate	Hours	Total Cost
Richard Clarke	Principle Investigator	$ 43	1500	$ 64,500

Principle Engineer	Design	$ 60	1500	$ 90,000
Senior Software Eng. I	Design and Implementation	$ 30	4000	$ 120,000
Tony Kostichka	Graphical Interfaces, Comm.	$ 18	1500	$ 27,000
Junior Software En I.	Programming and Support	$ 14	4000	$ 56,000
Syd Cherney	Business Management	$ 20	200	$ 4,000
Steven Burnett	Technical Writing	$ 13	400	$ 5,200
			Total Direct Labor:	**$ 366,700**

13.10. Labor Overhead

Labor Overhead Rate: 24%
Hour Base: 13,100, $ 366,700

Total Labor Overhead Cost: $ 88,008

13.11. Special testing: N/A

13.12. Special Equipment: N/A

13.13. Travel:

The travel budget will include attending conferences, training sessions, meeting with consultants and working on-site with customers on demonstration applications. Specific travel plans cannot be made at this point, but the following list provides estimates of the number of trips (by individuals) and expected expenses. Cost per trip is estimated using per diem of $150 plus estimated transportation cost.

Number of Trips	Purpose of Trip	Estimated Cost/Trip	Total
8	Attending Conferences related to the work effort. Estimate is based on two individuals attending four four-day conferences each, including conference registration costs.	$ 1,600	$ 12,800
4	Meeting with Consultants	$ 700	$ 2,800
4	On-site work on demo projects. Note that each trip is estimated at one week.	$ 1,500	$ 6,000
		Total Travel Budget:	**$ 21.600**

13.14. Consultants

Consultants will be used from the offices of the following three university professors, for work related to the indicated software systems developed under the direction of those individuals. The consultants will be either the professor or graduate students or research staff as available and as recommended by the professor. Consulting rates will be $ 60 per hour.

Professor, Location and Work	Hours	Total Cost
1) Dr. James Purtilo, University of Maryland For work on the Polylith software system.	100	$ 6,000

2) Dr. Jane Liu, University of Illinois For work on the PERTS software system	40	$ 2,400
3) Dr. John Reif, Duke University For real-time prototyping language work.	200	$ 12,000
	Total Consulting Cost:	**$ 20,400**

13.15. Other Direct Costs: N/A

13.16. General and Administrative Overhead

G&A overhead is applied to the total of all the above costs, using a rate of 29%.

Total Expenses:	$ 527,708
G&A Overhead:	$ 153,035
Total Cost:	**$ 680,074**

13.17. Royalties: N/A

13.18. Fee or Profit: **$ 50,000**

13.19. Total Estimated Cost: **$ 730,743**

13.20. Authorized Signature:

Signed: _____ **Date:** October 19, 1993

Richard J. Clarke, President of RTware, Inc.

13.21. Answers to specified questions:

a) Has any executive agency of the United States Government performed any review of your accounts or records in connection with any other government prime contract or subcontract within the past twelve months? NO

b) Will you require the use of any government property in the performance of this contract? NO

c) Do you require government contract financing to perform this proposed contract? YES

13.22. Type of Contract Proposed: **Firm-fixed price**

Bibliography

[Aonuma 1987] T. Aonuma, *An interactive Simulation Modeling System: DYNAGRAPH for multi-period planning on an APL spreadsheet*, Kobe University of Commerce, Kobe, Japan (1987).

[Binns 1993] P. Binns & S. Vestal, Formal Real-Time Architecture Specification and Analysis, to appear *IEEE Transactions on Real-Time Operating Software and Systems* (1993).

[Clarke 1991] Clarke, R.J., Real-Time Control: The Spreadsheet Paradigm. *Proceeding of the Tenth Annual Control Engineering Conference*, (May 1991) pp. 143-150.

[Freiser 1992] H. Freiser, *Concepts and Calculations in Analytical Chemistry, a Spreadsheet Approach*, CRC Press, Boca Raton, FL (1992).

[Hof 1990] Hofmeister, Atlee and Purtilo, Writing Distributed Programs in Polylith, University of Maryland Computer Science Department, Nov 1990.

[Kral 1992] I.H. Kral, *The Excel Spreadsheet for Engineers and Scientists*, Prentice-Hall, Englewood Cliffs, NJ (1992).

[Liu et. al. 1993] J. Liu, PERTS: A Prototyping Environment for Real-Time Systems, Report No UIUCDCS-R-93-1802, University of Illinois at Urbana-Champaign (1993).

[MN+92a] "Prototyping N-body Simulation in Proteus", P. Mills, L. Nyland, J. Prins and J. Reif. Proc. of 6th International Parallel Processing Symposium, Beverly Hills, CA, pp.476-482, IEEE, March 1992.

[MN+92b] "Prototyping High-Performance Parallel Computing Applications in Proteus", P. Mills, L. Nyland, J. Prins, J. Reif, Proc. DARPA Software Technology Conference, Los Angeles, CA, April 1992, DARPA, pp 433-442, April 1992.

[MPR93] "Rate Control as a Language Construct for Parallel and Distributed Programming", P. Mills, J. Prins, and J. Reif. Proc. of IEEE Workshop on Parallel and Distributed Real-Time Systems (IPPS'93), Newport Beach, CA, April 1993.

[Morrow 1991] V. Morrow, *Handbook of financial analysis for corporate managers: with spreadsheet models for decision analysis and performance evaluation applications*, Prentice-Hall (1991).

[Purtilo 1991] Purtilo, J., The Polylith Software Bus. To appear, *ACM Transactions on Programming Languages and Systems*. Currently available as University of Maryland TR-2469

[Purtilo 1985] Purtilo, J. Polylith: an Environment to Support Management of Tool Interfaces. *ACM SIGPLAN Symposium on Language Issues in Programming Environments*, (July 1985), pp. 12-18.

[Purtilo et. al. 1991] Purtilo, J. D. Reed, D. Grunwald, Environments for Prototyping Parallel Algorithms. *Journal of Parallel and Distributed Computing*, vol. 5 (1988) pp. 421-443.

[Redondo 1993] J.L. Redondo, Schedulability Analyzer Tool, Report No. UIUCDCS-R-93-1791, University of Illinois at Urbana-Champaign (1993).

[Schel 1991] C. Schedlberg, The Problem with Large, Tightly Coupled, Distributed Control Systems and How to Solve It. *Proceeding of the Tenth Annual Control Engineering Conference* (1991) pp. 161-171

[Sil93] A. Silberman, "Task Graph Model", in RTM: an Object-Oriented, Data-Driven Real-Time Environment," Ph.D. thesis (in preparation), Department of Computer Science, University of Illinois at Urbana-Champaign, 1993.

[Snodgrass & Shannon 1989] Snodgrass, R. and K.P. Shannon. Mapping the Interface Description Language into C. *IEEE Transactions on Software Engineering*, vol. 15, no. 11 (1989) pp. 1333-1346.

[Snodgrass 1989] Snodgrass, R., *The Interface Description Language, Definition and Use*. Computer Science Press, Rockville, MD (1989)

ControlCalc User's Guide and Reference Manual, RTware Inc., Durham NC (1991)

Labview for Windows User's Guide, National Instruments, Austin, TX (1993)

MetaH Language Reference Manual, Honeywell Systems and Research Center, Minneapolis, MN (1993)