Open Tool Kit for Mission-Critical Systems

A Phase II Submission to Navy SBIR N01-79[1]
by The Open Group

A. Abstract

This project will define, prototype and evaluate a framework and tool kit for building distributed real-time, fault-tolerant, mission-critical systems in heterogeneous environments. The primary elements are: CORDS/GIPC, a group communications system that provides fault tolerance in scalable, distributed real-time system; and a Real-Time Java environment for hard real-time applications. Other elements of the tool kit include: benchmarking and test control. The targeted commercial product would address both the initial development and the continuing development and life-cycle maintenance of evolving mission-critical applications, which would be written primarily, but not exclusively, in Java. This project will also initiate a standardization effort based upon the interfaces defined for the framework.

B. Summary of Phase I Results

As part of the Phase I work, we explored the development of a product to enable fault-tolerance within distributed, real-time systems using the Java language. This effort investigated the use of recent technology to address the needs of distributed real-time, fault-tolerant, mission-critical systems in heterogeneous environments (referred to as mission-critical systems herein). Our emphasis was oriented towards the interaction of two particular components: The Open Group's CORDS/GIPC[2] group communication protocol framework and tool kit; and Real-Time Java Virtual Machines (JVMs) based on the recently developed Real-Time Specification for Java (RTSJ)[3].

Our analysis and some demonstration experiments indicate that CORDS/GIPC and Real-Time Java JVMs together can effectively address the three problem areas that were emphasized in the original SBIR solicitation (N01-079) and subsequent discussions: operation in heterogeneous environments, meeting hard real-time deadlines, and supporting fault-tolerant operation. Both components are portable and real-time-capable. The real-time JVMs support the "write-once, run anywhere — carefully" motto espoused by the RTSJ expert group. CORDS/GIPC provides an easily understood model for implementation of fault tolerant, real-time applications. Furthermore, each component is at an early stage in its development cycle and can be expected to improve significantly as the technology matures.

---

[1] The original document was submitted in February, 2002. This version has been minimally updated to eliminate proposed tasks that were not accepted and to eliminate boilerplate.

[2] http://www.opengroup.org/RI/technologies/gipc/

[3] http://www.rtj.org/

Concurrently, we discussed the use of such technology with a number of defense contractors and found a common theme: They have an interest in the Java and group communication capabilities — and in other evolving technologies — and are aware of the benefits that they would provide. But, these new technologies are unfamiliar and have not been proven by extensive use in real-time, fault-tolerant, mission-critical systems. One recent surprise was that the estimated cost for the initial "COTS refresh cycle" for the Aegis Weapons System turned out to approach the original deployment cost of the system. Thus, they approach these new technologies warily and are very desirous of tools to assist them in understanding the use of such new technologies in their mission-critical systems.

We determined that there was a larger issue, which we also discussed with these users. Their interest in these products, and other commercial products, derives from a goal to develop more comprehensive, network-centric weapons systems. They want to build systems that are more flexible, more adaptable, and more maintainable. They are being encouraged to build components that can be reused in other systems, which would then support other, diverse missions. In short, they want to abandon stove-piped systems.

There is no corresponding reduction in system requirements, however. The systems must still satisfy stringent performance goals, including handling increased numbers of targets with higher availability in hard real-time environments. In addition, these flexible, adaptable, highly configurable systems must continue to be certified to perform properly in field conditions.

We believe, and have confirmed with potential users, that this larger need can best be met by extending the goals of the project to provide a common development environment based on a flexible framework. This framework would be populated with a tool kit that includes the Real-Time Java and CORDS/GIPC components from Phase I, as well as other, commercially available or privately developed tools that aided development of these mission-critical systems.

We also determined that there is a similar need in the commercial world. Although businesses do not normally encounter the same certification requirements, the difference is one of formality rather than necessity. Failure of mission-critical systems in the business world can cause huge monetary losses, so the commercial world shares an interest in ensuring that its systems will behave properly, maintaining stable, predictable services even in previously untested configurations. Potential customers include telecommunications, factory automation, refinery operations, etc.

C. Phase II Technical Objective

The overall goal of the Phase II effort is to define, prototype and evaluate a framework and tool kit for building distributed real-time, fault-tolerant, mission-critical systems in heterogeneous environments. This tool kit will provide commercially available alternatives for many functions that are often purpose-built for each systems, including a real-time group communication protocol to support distributed fault tolerance, a Real-

Time Java environment, synchronized clocks, and a coherent interface for system control and instrumentation.

This effort will produce prototypes of several related products:

- A specification of the extensible open tool kit framework
- A framework implementation to support tool kit components
- GIPC, a real-time group communication protocol implementation
- CORDS, a real-time-capable communication framework and protocol tool kit
- A Real-Time Java JVM adapted for use in the framework
- A set of Java objects for supporting real-time applications within the framework environment
- A test controller and harness for managing system tests and benchmarks
- “Wrappers" for several COTS and/or open source components so that those components operate as tool kit components
- A set of template "wrapper" functions to assist application developers in adding additional COTS and/or reusable components into the tool kit environment.

In addition to the overall goal of creating prototypes of an eventual commercial product, the Phase II effort will produce a functional tool kit that is suitable for use during development of prototypes of mission critical systems. This capability will be demonstrated via ongoing collaboration with developers of such system.

C.1. Frameworks and Tool Kits

Given the turmoil in word usage within the computer industry, it is useful for us to define our usage of these terms. For the purposes of this proposal, a framework is "a skeletal or structural frame," and a frame is "the constructional system that gives shape or strength" and "such a skeleton not filled in." Thus, we view a framework as a set of common interfaces and the structure for connecting components using those interfaces. Then a tool kit comprises that framework, a set of components that adhere to those interfaces and can populate the framework, and a set of auxiliary components that support the development and use of the framework.

The framework proposed here provides common interfaces for functions commonly used in the targeted systems. This simplifies the task of developing applications for the system, resulting in a lower need for continually retraining of developers and a subsequent reduction in development errors. Many suppliers of highly portable commercial software

incorporate so-called "glue" layers to create a virtual operating environment for their software. For example, Oracle has a special environment. Many vendors utilize POSIX or UNIX standards for this purpose. The ACE and TAO environments[4] are well known in the open source community.

As a specific example, consider the device driver framework in UNIX operating systems. At the most fundamental level, the interface to device drivers consists of only a few basic interfaces: *open*, *close*, *read*, *write*. Many, many utility programs operate using only those interfaces. Examples include, sorting programs, printer drivers, archival formatters. At a higher level of complexity, programs require specialization. Most applications require that certain devices provide a storage capability, that is, they require the *lseek* function and expect that a byte that is written at a particular offset can later be retrieved by reading from that same location. On the other hand, many of those same applications also require that certain devices behave as TTYs, i.e. that there is a visual relationship between successive bytes. (Consider displaying an "ASCII" document, whereby it is assumed that successively written bytes are displayed in rightward succession across a page——as modified by formatting characters, such as BS and NL, of course.)

The power of frameworks, in general, and the device driver framework, as a particular instance, comes from the conformity of the constituent components. Many application developers write large, complex applications without having to worry about what kind of disks are operating behind the standardized system interface. Whether the disk is SCSI-I, SCSI-II, or even ATA doesn't matter. In fact, these lower level standards allow even operating systems to function without having to know whether their disks were manufactured by Maxtor or Western Digital. Conversely, an embedded system in a harsh environment might employ a "solid-state disk," which isn't a disk at all, but rather a RAM-based device that adheres to the interface standards defined for disks. Because this device appears to be a disk, it will function properly with all of those thousands of programs whose function complies with the disk interface standards.

It is this power that we intend to bring to real-time, fault-tolerant, mission-critical systems. Consider the instrumentation aspect of these systems. Most real-time systems interact at some point with real-world, physical elements and must meet the time constraints imposed by the laws of physics. Use of "breakpoints" from interactive debuggers generally is not practical because it interferes with meeting those time constraints. As a result, debugging in many real-time systems is performed using event tracing. In this approach, software probes log event information into a common log. The developer can perform a post mortem analysis of this log to determine the order in which events occurred. By utilizing information included in the event log entries, the developer can isolate a problem. Note that it is often the ability to determine that event A occurred before event B that is important to determining the root cause. Thus, the use of a common log is crucial because entry of information into this shared resource enforces a total ordering of events. Use of a common log requires that all participating components

[4] http://ace.cs.wustl.edu

adhere to a common interface for logging. The power of a common framework is that if a newly installed application uses that common logging interface, its interactions with existing applications can be more easily debugged.

Consider the extension of this tracing mechanism to distributed systems, where it is not possible to introduce a common log without interfering with system time constraints. One viable solution here is to introduce the notion of synchronized clocks internal to the logging facility. This allows events to be accumulated on each computer node independently. These separate logs can then be merged post mortem based on time stamps that are generated for each entry. The verity of ordering within the merged logs then depends only on the precision to which the clocks are synchronized. The power of the interface here is that it was only the logging facility itself that needed to be upgraded for use in the distributed environment. All applications that complied with the original, local-only interface automatically received the benefits of the improved logging facility.

The differential for this framework is that it will address the issues in building distributed real-time, fault-tolerant, mission-critical systems. Thus, the framework will enable the functions associated with such systems, including managing timing constraints, detection and recovery from various failures, and dynamic reconfiguration based upon environmental changes. At the same time, these systems share much in common with other systems, so the framework will incorporate commercial standards where possible, in order to offer the greatest probability that COTS components will adhere to it.

C.2. Interface Definition

The initial definition of the framework will be based on interfaces that have been identified by The Open Group during many years of working with the target community through our research programs and through Forums, such as the Real-Time and Embedded Systems Forum[5] and the Quality of Service Task Force[6]. These interfaces reflect the needs of both application system developers and product vendors in the field. Early versions of particular elements have appeared in Open Group products, including the CORDS communication framework, the GIPC group communication protocol system, and the MK 7 real-time operating system.

The Phase II work, however, envisions a far more comprehensive framework than currently exists. As additional components, such as Real-Time CORBA, are brought in, the interfaces in the framework will very likely require slight modifications to increase the usability of the overall tool kit. In addition, a number of standards have evolved and will continue to evolve in this area. Therefore, we will conduct an iterative process: define, prototype, evaluate, repeat.

---

[5] http://www.opengroup.org/rtforum/

[6] http://www.opengroup.org/qos/

C.3. Prototyping

There is a gulf between the expectations of the developers of a product and the users of that product. The developers are technical experts in the area and well versed in the trade-offs among various alternatives. In the case of component products as contemplated here, the users are also technical experts — but usually in a different area. These users have selected the product in order to simplify their overall development effort. This clash results in user confusion and misunderstanding — and often in a misapplication, or at least underutilization of the product. To alleviate this problem, we have identified several prospective users of this product and will work with them to improve the effectiveness of the product.

We will create several iterations of various components within the tool kit. Each successive prototype will provide an increased level of function as well as incorporate lessons learned from previous versions. This staging of deliveries decreases the amount of development effort required for initial delivery, thereby increasing the amount of customer exposure to the product and reducing the amount of rework needed when product specification changes are required.

C.4. Evaluation

We expect the most productive form of interaction with these users to be as part of their product evaluation cycle. All of these organizations evaluate new products as an explicit activity within their development methodology. They create technology evaluation teams that perform increasingly realistic (and increasingly expensive) tests. Starting with reviews of research papers and product literature, these groups progress to simple experiments, and then to benchmarks, and eventually to subsystem tests. The evaluation teams are also the basis by which the organizations acquire the corporate knowledge about how to apply the technologies.

C.5. Iteration

We anticipate working with the prospective users via these technology evaluation teams on an ongoing basis. We will assist them in learning about the technologies included in the tool kit. In return, we will acquire valuable assistance in adapting our proposed product to the needs of real users. In addition, we will gain insight into subtle difficulties that might arise between our product and other components with which the users must interoperate.

As the users gain confidence in the performance of our prototypes, they will become more comfortable in including the expected product within their mission-critical systems. In addition, we will gain more insight into additional opportunities for applying and extending the tool kit.

C.6. Standardization

In the past, real-time, fault-tolerant, mission-critical systems used defined their own marketplace. Each system would be specially designed and most components were purpose-built for use in a particular system. The (intended) result was a system with an overall, unifying architecture. Individual subcomponents interacted with other subcomponents via well-tuned interfaces and every subcomponent provided required performance data via a system-wide instrumentation system. In short, these systems were homogeneous and effective.

The design of these systems was, however, brittle —stove-piped in the vernacular. Replacement of components required significant rework, so many component upgrades were structured to emulate the original component. More importantly, these systems were not extensible. It was difficult to incorporate new capabilities and to interface to additional systems. This difficulty was expressed as both high cost and long turn around times for functional upgrades. One notable example is the use of FDDI in Navy ships. When commercial manufacturers terminated production of FDDI chips, a significant amount of software and hardware had to be reworked. The result of these types of problems has been a push towards reducing the use of purpose-built components. When possible, commercial, off-the-shelf (COTS) products are utilized. When COTS products are not available or not suitable, there is a desire to reuse components from other projects.

One effect of the increasing utilization of COTS components in mission-critical systems is that these systems are no longer able to specify the characteristics of the components. They have become dependent upon a component supply chain where the requirements are largely driven by the needs of a much larger marketplace —one that does not have the special needs of real-time, fault-tolerant, mission-critical systems. Consequently, the mission-critical market segment is attempting to draw from the capabilities targeted at the general marketplace. As a result, we now have the real-time additions to the POSIX specification, the Real-Time CORBA enhancements, and recently the Real-Time Specification for Java.

Another important aspect of the use of commercial products and of reusing internal components has been the introduction of significant heterogeneity in both the hardware and software components. The various components used in the system evolve based on external changes. It is no longer possible to require all new components to adhere to historic, system-specific interface requirements. Thus, there is a need for common interfaces within the system infrastructure. Components can be programmed to match those interfaces rather than the peculiarities of whatever component happened to be built at some point in the distant past. Use of such common interfaces also simplifies the maintenance cycle. Components will change, and the interfaces must evolve, but the cost and impact of these changes can be limited via explicit management of changes to the interfaces.

We recognize these effects and will address them as part of Phase II. We will work to make the proposed framework as standards-compliant as possible while still meeting the needs of the target customers. We expect to incorporate existing standards, such as

POSIX, Real-Time CORBA, and RTSJ. We will also track and attempt to influence evolving standards, such as the anticipated reliable multicast specification for CORBA, and the Common Information Model (CIM). Finally, it should be noted that The Open Group itself is a standards body, and we will work through our existing forums and task forces to generate consensus towards the establishment of any additional standards that might be required.

D. Phase II Work Plan

The development methodology within the research group at The Open Group involves two distinct phases: a research phase whereby new, advanced concepts are developed, prototyped and analyzed; and a technology transfer phase in which successful research projects are transitioned to external groups via collaborative projects where the technology is iteratively evaluated and improved. Part of the benefit of the proposed framework and open tool kit is that it is extensible and could address many elements of the computing infrastructure within mission-critical systems. We view a substantial portion of the Phase II work as a technology transition effort. The nature of mission-critical systems is that they must be dependable: they must be highly available, and they must operate correctly. Our intent is to concentrate on making selected elements of the proposed Open Tool Kit work well rather than to develop lots of elements.

There are two major themes in creating dependable software: deliberate development and insightful testing. While we expect to do the implementation work and initial testing, our potential customers are much better equipped to provide system level testing. They establish test-beds with realistic load simulation facilities as a normal part of their development process. To induce them into expending resources to evaluate and test our prototypes, however, we must maintain their interest by providing tools that address their actual problems. The result is that the selection of tool kit capabilities will be largely driven by the interests of our collaborators. If their interests change, presumably due to evolving requirements, we expect to adapt the tool kit to their altered needs. Nonetheless, we present here a spectrum of potential development concepts based on our best understanding of the needs of the target customers. This list is intended to convey the potential range of the proposed tool kit. Resources to completely implement this list would far exceed the expected available resources. Actual projects to be prototyped will be selected after consultation with the Technical Point of Contact and the technology transfer targets.

D.1. Framework and Tool Kit Architecture and Design

Based on our previous experience with real-time, fault-tolerant, mission-critical systems and with our discussions with our technology transfer targets, we have identified a number of areas for inclusion in the tool kit and thereby for specification of framework interfaces. The selection criteria included requirements to:

- Support distributed, real-time, mission-critical applications

- Interface to other systems, including other languages, hardware platforms, and distributed system technologies (e.g., Ada, C/C++, single board computers, Real-Time CORBA)

- Interface to development tools for real-time systems (e.g., scheduling tools, UML)

- Adhere to standards where possible (e.g., CORBA, POSIX, UNIX, Java)

The development of each capability area includes both tasks specific to that area and general tasks that must be done for all areas. The general tasks include evaluation of the underlying component, development of microbenchmarks, and definition of the actual framework interface specification for the area. Although we expect collaborators to provide advanced testing during the Phase II prototyping stage, elimination of most bugs must still occur prior to delivery to those collaborators. Thus, we will create initial test facilities for each area. Generally, these tests are also provided a part of the product because they demonstrate ways of using the components and of the framework interfaces. In addition, all of the internal and external products must be documented.

Although the use of external collaborators for advanced testing improves the effectiveness of the testing process, it also increases the need for early documentation and for external support. In many cases, we expect to need to conduct design reviews of customer applications and to develop additional testing software to isolate problems that customers may have detected.

D.2. Framework

A good framework imposes low system overhead while providing powerful capabilities through both native tools and third-party plug-ins. In essence, a good framework is elegant, providing its utility via effective access to the components that comprise the tool kit that it makes available. Thus, the framework itself comprises little code, leaving the bulk of its capabilities to the tool kit — the components that easily fit into the framework or tools that utilize those components.

Nonetheless, the framework must operate coherently and its constituent elements should not interfere with each other. For example, the time format returned from the distributed time components should be compatible with the time format used within system control and instrumentation. To evaluate the integrity of the overall system, we will develop a prototype application that exercises all areas of the framework and tool kit. Although the applications algorithms will be minimal, this application must respond properly to control and status requests, and it must provide accurate instrumentation data based on actual consumption of resources. We expect much of this work to be performed as part of the development of the Controller portion of the framework, and in conjunction with development of TET controller component.

The Open Tool Kit Framework must operate in some operational context. Based on the interests of our technology transfer partners, we anticipate using Real-Time CORBA for many of the control functions. We will select a one or more Real-Time CORBA implementations in conjunction with the partners. TAO[7], an open source implementation is an interesting candidate. We may also need to use a commercial product, such as ORBExpress from Objective Interface Systems.

D.3. Component Acquisition and Development

We provide here a summary of the tasks that are expected for various components that we expect to include in the initial tool kit. There are a few elements of the framework that are crucial to the utility of the overall concept. These include control/status interfaces and provision of instrumentation data. These aspects must be addressed for every component that is to be included in the tool kit, and the implementation of the supporting software is not called out separately below.

D.3.1. GIPC/Group Communications

Group communications is a model of communicating between multiple computers. Simplistically described, group communications provides the ability to do an atomic broadcast within a group. Such an atomic broadcast in a distributed environment can be compared to a transaction for traditional DBMSs. A group is a set of cooperating processes that want to communicate with each other. An atomically broadcast message is directed toward a group, and is either delivered to all group members or none. Applications can then leverage this basic function in order to easily implement application-specific fault tolerant strategies.

One of the earlier implementations of group communications was the ISIS system, which was developed at Cornell University. ISIS was spun off into a commercial company and was widely used on Wall Street for financial transactions systems. The commercial product appears to have been discontinued. Cornell later developed Ensemble, a research system similar to ISIS with additional capabilities. Ensemble is available for general use.

The Open Group has developed CORDS and GIPC. CORDS is a general purpose framework for developing communication protocols. GIPC is a Group IPC system that supports group communication. A distinguishing feature of CORDS and GIPC is that they were designed to support group communications in real-time applications. A prototype demonstration system that uses CORDS and GIPC to manipulate a ball sorting apparatus can detect and recover from a node failure in less than 400 msecs[8]. The GIPC (Group InterProcess Communications) package will be extended to include additional

---

[7] http://www.cs.wustl.edu/~schmidt/TAO.html

[8] L.M. Feeney, P. Bernadat, F. Travostino, Characterizing Group Communication Middleware for Real-time Distributed Systems, Work in Progress Report, 18th IEEE Real-Time Systems Symposium, December, 1997, San Francisco, CA.

protocol support and to create development support tools. Several areas have been identified as useful enhancements, including an interface to external failure detectors, such as our Fast Failure Detector component, explicit control over use of redundant network attachments, and access to QoS-related network parameters. We may also want to add specialized support for use of GIPC from a Real-Time Java environment.

D.3.2. Real-Time Java

A fundamental part of the strategy is the use of a real-time version of Java. Java provides an execution environment that is largely independent of both the underlying hardware platform and of the operating system. A Java class object is machine independent: the exact same executable object can run on any Java machine. Each object runs in the context of a virtual machine that provides identical behavior for almost all machine characteristics, including integer ranges. Java is becoming very popular, and Java products now exist for many environments, ranging from constrained embedded systems, such as set-top boxes, to World Wide Web server applications.

The creation of a version of Java capable of supporting real-time applications with specified time constraints was the first enhancement officially proposed for Sun's Java. The result was recently published and several implementations of the real-time specification are currently under development by members of the specification expert group. The reference implementation, which was developed by TimeSys, has just been released in its completed form.

Real-time Java incorporates an innovative method for dealing with the garbage collection problem. Certain real-time threads can be declared to be higher priority than the garbage collector and can, therefore, defer the execution of the garbage collector. To prevent deadlock, these threads must allocate space only from certain reserved heaps, which are not subject to garbage collection. A method is then provided to allow information from the non-garbage-collected heaps to normal processing space. Although this method is highly promising, the ability of real-time application programmers to grasp the concepts and use it effectively remains to be proven. The recent approval of the Real-Time Specification for Java (RTSJ) is expected to presage the release of several commercial Real-Time JVMs. Most providers will be targeting traditional embedded applications. Thus, the primary extensions that are needed for this tool kit are to address real time and fault tolerance requirements in distributed systems. These include a package to interface to the group communication protocol, and a package to interface to a real-time publish/subscribe protocol. In both cases, we will be tracking efforts within OMG to develop related, CORBA-based specifications.

Based on their target marketplace, we expect the Real-Time JVM providers to provide efficient access to a operating platform's native TCP/IP protocol stack via the *java.net* interface. We anticipate the need, however, to develop improved and/or alternate packages for more structured access, such as is provided by the *java.rmi* package, which implements an RPC-like mechanism. (The problem here is that RTSJ supports real-time applications by providing an alternate memory management scheme to traditional

garbage collection. Java's RMI subsystem was previously developed and the standard implementation may not comply with the restrictions required by RTSJ.) We also expect that we will need to provide access to newer, more real-time-capable transport protocols, such as the SCTP protocol mentioned above under GIPC.

D.3.3. TET (Test Environment Toolkit)

TET[9] is an example of the leverage that can be gained by effective use of the open source world. TET was originally developed by the Open Software Foundation and X/Open (whose merger formed The Open Group), and by UNIX International as a framework and tool kit for system testing. As a group, these three organizations — and their constituent membership — virtually controlled the specification of the UNIX API (application programming interface). They both defined the interfaces and provided the compliance test suites for UNIX. Based on its wide applicability and its freely available status, the membership organizations made broad use of TET as a basis for creating additional test suites. Today TET is widely used throughout the industry, although the tests are mostly used for internal development so its visibility is relatively low.

Recognizing the potential market represented by this broad use, the testing and certification organization within The Open Group has created TETware professional. As the name suggests TETware professional is a commercial product, providing additional capabilities, automated installation, and committed support. The Open Group continues, however, to maintain the freely available version of TET in order to increase the potential market size.

We expect to use TET as a component for controlling a system in a test environment. We would like to use the freely available version of TET in a basic version of the framework. That would allow us to provide a basic operating capability in a freely available version of the Open Tool Kit (see Commercialization Strategy). We may also need to provide extensions to the TETware professional product if our technology transfer partners indicate a need for its use in their test beds. It is also possible that our technology transfer partners are already using another test environment controller, in which case we would assist them in hooking that controller into our framework.

D.3.4. CIM (Common Information Model)/Pegasus

The Common Information Model (CIM) is an object-oriented information model that presents a common representation of data in a "managed environment." The Distributed Management Task Force (DMTF)[10] and The Open Group Enterprise Management Forum[11] are working cooperatively to manage this evolving standard. The DMTF is

---

[9] http://tetworks.opengroup.org/

[10] http://dmtf.org/

[11] http://www.opengroup.org/management/

continuing to develop and expand the models, schemas and interoperability characteristics of the CIM model while The Open Group is concentrating on providing an open source implementation of these concepts and extending the concepts to areas of manageability such as management of applications, services, and Quality of Service.

CIM comprises three layers: the core model, which is applicable to all areas of system management; the common model, which addresses particular application areas; and extension schemata, which are specific to operational environments. Although CIM originated in the world of mainframe application management, we believe that the concept can be extended to the realm of mission-critical systems. We would work to develop a new common model that would specify the information associated with real time and dependability as well as an extension schema based on our framework. These schemata would include definitions, such as coherent definitions of message latency and time base, as well as the object classes, properties, methods and associations that are needed to manipulate those values.

Although the consistency provided by the underlying schemata provides a useful base to enable interoperability of tool kit components, there is also the potential to include a run time CIMOM (CIM Object Manager). There are several implementations currently underway, some of which are open source. One of particular interest is Pegasus, which is an open source implementation initiated by The Open Group's Enterprise Management group. Release 1.0[12] was recently distributed under the MIT license. HP, IBM, and Compaq have all indicated that they intend to adopt the Pegasus reference implementation for use within their product lines.

D.4. Framework Development

As noted earlier, much of the development for the framework will be performed in conjunction with the development and/or adaptation of components that will populate the various framework areas. It is useful, however, to associate development goals with framework areas.

D.4.1. System Management/Controller

The System Manager provides control over the state of the system. During the Phase II development effort, we anticipate that the primary use of this function will be to control benchmarks and debugging sessions. In addition, this framework role needs to be populated in order to exercise the control/status interfaces (see below). We expect to use TET (see above) for this purpose.

We also expect to identify components useful for benchmarking and testing. Typical components would include network and CPU load stress generators and fault insertion tools.

---

[12] http://www.opengroup.org/pegasus/

D.4.2. Control/Status Interface

Every system requires some amount of control function, even if that function is limited to creation and destruction of the processes that implement the system. As systems become more complex, it becomes more useful to provide abstractions. Thus, it is useful to consider an operator initiating the weapons launch function rather than having the operator understand the that he must start process A on host B, and then 5 seconds later start process C on host D, but if host D is down, then start process E on host F, ...

Development of new, distributed weapons systems, such as Aegis Open Architecture, will require the reimplementation or significant rework of many applications. It is a given that the developers will include some form of control and status interface. Early definition of standard interfaces for control and status will foster a more cohesive, more coherent system. We intend to address this area very early in the Phase II effort.

D.4.3. Instrumentation/Visualization

Earlier in this proposal, we described the extensive use of event tracing for debugging real-time systems. Almost all such systems also include mechanisms for gathering performance data and displaying the data for analysis by resource managers and/or humans. The situation with instrumentation and visualization is similar to that for status and control: the application developers will be doing something, so early definition of interfaces will foster a more cohesive, more coherent system. Again, we intend to address this area very early in the Phase II effort.

There is a major difference from status and control, however, in that instrumentation requires a component to gather, log, and redistribute the instrumentation data. We have identified several candidates for this function. System/Technology Development Corporation is developing QMS (QoS Metrics Services)[13]. QMS is interesting because it is designed to be compatible with CIM (see above). On the other hand, QMS is based on use of CORBA event channels, so the per-message overhead may be too high. Another possibility it to build upon the Linux Event Logging For Enterprise Class Systems[14] facility. This component has the dual benefits of attempting to conform to the draft POSIX SRASS (Services for Reliable, Available and Serviceable Systems) standard and of being adopted by IBM for use in enterprise systems. One major drawback is that it is targeted at enterprise applications, so its use in real-time systems may be problematic.

We will consult early in the Phase II effort with the Technical Point of Contact and our technology transfer targets to decide a plan of action for this area.

---

[13] http://www.stdc.com/QMS/

[14] http://evlog.sourceforge.net/

D.4.4. Interfacing to Tools

Design tools are becoming more powerful and consequently more useful in real-time systems. For example, Aegis Open Architecture includes Rational Rose as part of the development. I-Logix's Rhapsody environment provides similar capabilities. Consequently, we must plan to develop UML models for (at least) the framework components that will be utilized directly in their systems.

Use of effective modeling tools for designed mission-critical system would provide real benefits in reducing complexity and in generating the information needed for effective management of system resources during dynamic operation. Although the first generation of "real-time" UML tools failed even to include any capability for actually dealing with time, the second generation of tools from Rational Software[15], Tri-Pacific Software[16], TimeSys[17], and other vendors appears to be useful for statically defined systems using rate monotonic scheduling. Although rate monotonic scheduling is not likely to be extensively utilized in complex systems such as the Aegis Open Architecture, these tools represent real progress and offer much promise for future capabilities. We would like to include the capability for framework components to interact with the tools in providing information to the resource manager.

D.5. Core Facilities

There are a number of enhancements that may be needed in the core facilities of the framework. We plan to structure this development into two major phases with milestones:

- SOC +12 months: Initial delivery of framework using Real-Time CORBA; benchmarking tools; CORDS/GIPC support tools
- SOC + 20 months: complete framework, including core facilities; documentation, test suite.

D.5.1. CORDS

There are two tasks related to the basic CORDS framework that would potentially be of use to GIPC. We will investigate the implementation of the Stream Control Transport Protocol (SCTP)[18], which is a recent addition to the TCP/IP suite that provides better control to real-time-aware applications. In addition, we will explore the utility of porting the CORDS communication framework (and thus the GIPC module) so that it executes within the context of a loadable device driver for a real-time operating system.

---

[15] http://www.rational.com

[16] http://www.tripac.com

[17] http://www.timesys.com/

[18] ftp://ftp.isi.edu/in-notes/rfc2960.txt

D.5.2. Distributed Clock Management

The NTP[19] (Network Time Protocol) component is ubiquitous within systems requiring clock synchronization across multiple network nodes. Although the software is open source, the internal algorithms are effectively incomprehensible other than to members of the NTP project. Reflecting improvements, tweaks, and just plain hacks, these algorithms reflect experience gained over many years of operation within the Internet environment, and the use of NTP provides many benefits. The policies incorporated within NTP are oriented toward defense from "false tickers" and denial-of-service attacks and are not oriented towards the special needs of real-time, mission-critical systems. As an example, NTP will tolerate a significant difference (more than a tenth of a second) between clocks on different nodes. Also, NTP time provides its own definition of time (that is, NTP time does not correspond to TAI, UTC, UT0, or any other established standard).

Nonetheless, it is likely that NTP is the most useful base component available for providing a synchronized clock capability within a real-time, mission-critical system. One addition that we expect to include within the Open Tool Kit framework is an estimate of the error incorporated in the time available on the local system.

D.5.3. Failure Management

We anticipate working with a major defense contractor on fault management strategies within real-time, fault-tolerant mission-critical systems. We have not yet, however, selected a particular application for consideration, so we do not yet have a specific plan of action. We have, however, been discussing the use of The Open Group's Fast Failure Detector (FFD), which was developed as part of the DARPA-sponsored Quorum program. The HiPer-D project at NSWC Dahlgren has demonstrated that FFD, in conjunction with a real-time operating system, is capable of reliably detecting failed computer nodes in less than 200 milliseconds, even in the presence of a significant CPU load. We expect to include FFD in the Open Tool Kit framework and provide access to it via the control and status interface.

D.6. Support of Technology Transfer Targets

There are a number of activities that will directly address collaborative activities with the technology transfer targets. The most visible is direct support of use of the Open Tool Kit within weapons systems applications. Other relevant tasks include: early documentation of individual framework components and the associated interfaces, and creation of example worked cases to demonstrate the use of the overall system.

We have identified three technology transfer targets with whom we intend to collaborate during Phase II. All have expressed interest in issues that permeate infrastructure development, such as managing time constraints across distributed systems, maintaining

[19] http://www.ntp.org/

synchronized clocks, instrumentation, etc. We will work with these organizations in applying the tool kit components to their overall problem. In addition, each group has identified particular issues:

We have worked for several years with the HiPer-D project at the Naval Surface Warfare Center (NSWC) in Dahlgren, VA, which investigates advanced technology on behalf of the Navy and of Navy contractors. This group verified the utility of The Open Group's Fast Failure Detector (FFD) recently, but it was unable to demonstrate the capability due to problems with their applications, which were using the Ensemble research group communication system — a non-real-time-capable system. We have proposed that we will work with the HiPer-D project in applying CORDS/GIPC to successively "hard" real-time applications.

The required tasks for support of HiPer-D can be more precisely defined and we have identified associated milestones. We anticipate two deliveries for HiPer-D:

- June, 2002: Delivery of prototype CORDS/GIPC/RT-JVM suitable for use in Demo 2002 (September 2002) (“soft” real-time).

- June, 2003: Delivery of prototype framework with CORDS/GIPC/RT-JVM suitable for use in Demo 2003 (September 2003) (“hard” real-time)

D.7. Standardization

For customers of this technology to derive full benefit from its use as an integration tool, it must fit in smoothly with both the overall system architecture and the component being integrated. To achieve success in the COTS marketplace, the technology must align with emerging standards, such as those for object-oriented design for mission-critical and real-time systems. We have identified several relevant industry standardization activities that we would participate in as part of Phase II:

- The Distributed Real-Time Specification for Java (DRTSJ)[20] Experts Group, operating as part of the Java Community Process sponsored by Sun, will establish standards for key capabilities used in this environment, such as access to real-time (and "real-fast") networking facilities within real-time Java. The Principal Investigator for this contract, has been participating in this group as part of Phase I work, and will continue during Phase II.

- The Object Management Group (OMG)[21] is developing an extension to the CORBA standard for reliable, group-ordered multicast. It is also expected that this facility will be retrofitted to be used as a foundation for Fault Tolerant

[20] http://www.drtsj.org/

[21] http://www.omg.org/

CORBA, which provides highly-available CORBA objects in distributed object systems. The OMG is also shifting the CORBA standard's C++ focus to an implementation-independent approach as part of a move towards Model-Driven Architectures. The benefit of the proposed technology in integrating across COTS components is based on commonality between C++ and Java. We plan to work with the OMG specification authoring groups to provide as much commonality as possible between the Java and C++ approaches.

- The Open Group's Real-Time and Embedded Systems Forum[22] conjoins system vendors and end customers to develop standards and certification programs that speed market uptake of real-time and embedded technologies. Recently, the group has focused its efforts on establishing standards and certification programs in support of high reliability and safety critical systems. The high level of participation by commercial systems vendors makes this an ideal venue for transferring the proposed technology to the COTS marketplace.

D.8. System Integration, Test and Evaluation (OPTION)

The Phase II plan for system integration, test, and evaluation has three goals:

- Ensure the relevance of the proposed product to both military and civilian applications.

- Bring the proposed product up to date relative to marketplace shifts and newly available technologies.

- Apply the lessons learned from the collaboration with systems developers during the Phase II effort.

The following complementary and parallel activities will address those goals:

- Improvements to the CORDS/GIPC protocol component. The Phase II effort is intended to identify and define the improvements that are necessary to use CORDS/GIPC in hard real-time applications, particularly in conjunction with commercial networking gear. We will either make those improvements or create a plan for creating the improvements.

- Additional support tools for CORDS/GIPC. Use of CORDS/GIPC by the technology transfer target developers will suggest a set of support tools that will add to the usability of CORDS/GIPC and group communications by application developers. We will prototype those tools.

[22] http://www.opengroup.org/rtforum/

- The Distributed Real-Time Specification for Java (DRTSJ) process should have produced a specification during this period. We will investigate the changes that are needed to both CORDS/GIPC and the Real-Time Java environment objects and wrappers to determine if they should be updated and possibly prototype this capability.

- The anticipated OMG specification for reliable multicast should be relatively firm during this period. We will examine the possibility of adapting CORDS/GIPC in support of, or possibly to implement, this OMG specification and potentially prototype this capability.

- The potential for use of group communication in factory automation systems should be clearer in this time frame. We will investigate the applicability of CORDS/GIPC to this marketplace and potentially prototype this capability.

- Based on experience with the Open Tool Kit Framework, we will have identified desirable modifications to the interfaces and components. Many of these changes will have been deferred in order to obtain maximum benefit from the Phase II collaboration with systems developers. We will update the Open Tool Kit Framework specification based on this experience.

- Based on experience with the Open Tool Kit Framework, we will have identified additional system interfaces that are candidates for inclusion. We will investigate these additional interfaces, the incorporation of commercial components from other vendors in support of those interfaces, and the creation of new components to supply the functions. We will potentially prototype these wrappers and/or new components.

- We will continue to support the effort to create a standard based on the Open Tool Kit Framework interfaces.

E. Related Work

In addition to the previously cited components under consideration for the Open Tool Kit, The Open Group has been involved with a significant number of external projects whose focus is closely aligned with the goals of the Open Tool Kit:

The Adaptive Communication Environment (ACE)[23] provides an object-oriented (C++) framework for concurrent communications and an accompanying library of implementations which provide a common interface to diverse low-level operating system and network functions. In addition, implementations of common design patterns related to concurrency and network communications offer programmers higher-level

[23] http://www.cs.wustl.edu/~schmidt/ACE.html

abstractions to facilitate the design of concurrent applications. Commercial support for ACE is available from the Riverace Corporation[24]. The TAO ORB, which is built upon these common interfaces, was used extensively by The Open Group and its partners as part of its integration work for the QUITE project.

Globus[25] provides a framework, toolkit and user interfaces for Grid computing services over a wide-area network. Globus researchers at Argonne Labs and USC-ISI are currently working with IBM to evolve the Open Grid Services Architecture and to evolve the Globus toolkit to support this architecture. The Open Group, as part of the QUITE project, worked with the Globus team to integrate Globus resource management and directory services components with a local resource manager (based on the DeSiDeRaTa[26] project), and transitioned this integration to the HiPer-D test environment at the Naval Surface Warfare Center in Dahlgren, VA for inclusion in its Demo 2000.

Linux/RK[27] extends the Linux kernel to provide an abstract control interface for reserved access to operating system resources. The Resource Kernel technology has been incorporated into a commercial product, TimeSys Linux[28], by the TimeSys Corporation. The Open Group utilizes Linux/RK to provide resource guarantees for its Fast Failure Detector (see above).

Ensemble[29] provides a protocol library and development toolkit for writing virtually synchronous group communications applications. A predecessor project to Ensemble, ISIS, was transitioned to the commercial marketplace by ISIS Distributed Systems, which was acquired by Stratus Computers in 1993, and was commercially available until 1998. The Open Group utilizes Ensemble to provide group communications services for its Fast Failure Detector, which was recently transitioned to the HiPer-D test bed at the Naval Surface Warfare Center in Dahlgren, VA.

RTCAST[30] is a real-time group communications protocol which has been implemented atop The Open Group's CORDS communication framework and the MK 7.2 microkernel. Ideas developed in this project have been incorporated in commercial products under development by Arbor Networks[31].

---

[24] http://www.riverace.com

[25] http://www.globus.org

[26] http://desidrta.uta.edu/

[27] http://www-2.cs.cmu.edu/~rajkumar/linux-rk.html

[28] http://www.timesys.com/products/realtime/index.html

[29] http://www.cs.cornell.edu/Info/Projects/Ensemble

[30] http://www.eecs.umich.edu/RTCL/arpa-project/rtcast

[31] http://www.arbornetworks.com/

Cactus[32] provides a framework and several implementations (one, in C, based on The Open Group's CORDS communication framework and MK 7.3 microkernel) for customizable fine-grain QoS for secure, dependable, real-time systems. Research based on this work is ongoing at AT&T Research Labs[33].

Quorum[34] is a DARPA research program to investigate the applicability of QoS-based concepts to complex systems, in general, and to military weapons systems in particular. Much of The Open Group's interaction with the NSWC HiPer-D project has been sponsored under this effort.

QUITE[35] is a project to integrate many of the Quorum projects and apply the concepts to particular problems. The Open Group has been a major participant in this effort. One particular result has been the Fast Failure Detector that will be included in the tool kit.

There are also a number of projects within The Open Group that have provided a basis for many of the particular concepts in the Open Tool Kit that make it particularly applicable to real-time, fault-tolerant, mission-critical applications:

Many of the concepts used in real-time distributed systems were explored in the Alpha Operating System[36], which originated at Carnegie Mellon University. The principal investigator managed the creation of an Alpha prototype system while at Concurrent Computer. He later brought those concepts to The Open Group and merged them with the Mach Microkernel[37] from Carnegie Mellon University. The result was the MK 7[38] and AD operating systems. Many of these concepts were successfully transferred into commercial products that were produced by industry leaders, including Intel Supercomputing Systems, Hewlett Packard/Convex, IBM, Hitatchi, Honeywell Space Systems, and most recently Apple as the Mac OS X operating system.

The Alpha-based concepts are also appearing in non-operating system contexts, including: the paths concept in CORDS; in research projects, such as the DeSiDeRaTa resource manager described above; in the recently approved Real-Time CORBA 2 specification; and in the ongoing effort to create a Distributed Real-Time Specification for Java.

---

[32] http://www.cs.arizona.edu/cactus

[33] http://www.research.att.com

[34] http://www.darpa.mil/ito/research/quorum

[35] http://quite.teknowledge.com/quite

[36] http://www.real-time.org/papers/usenix92.pdf

[37] http://www-2.cs.cmu.edu/afs/cs.cmu.edu/project/mach/public/www/mach.html

[38] http://www.opengroup.org/RI/technologies/mk7/

Many of the communication-based framework concepts proposed for the Open Tool Kit are derived from CORDS[39]. CORDS, in turn, is derived from the x-kernel[40], which was originally developed by the University of Arizona. Implementations or concepts from CORDS have been used in many research projects, as noted above, as well as in products from both DASCOM and Novell.

F. Relationship with Future Research or Research and Development.

This Phase II effort will result in a set of middleware components and tools that will assist developers of real-time, fault-tolerant, mission-critical systems in building more effective, more maintainable systems. The most significant component is CORDS/GIPC, a group communication protocol implementation designed for real-time applications. With more than ten years of use in the financial community, group communications has proven to be an effective, usable model for implementing reliable fault tolerance. CORDS/GIPC will allow that experience base to be used in systems with hard real-time requirements.

This effort will also produce a set of components and tools for use in the computing infrastructure of these mission-critical systems. In addition to reducing development costs by replacing purpose-built components in the deployed system, these commercial components will also be designed to work together to simplify the overall development process, including component evaluation, benchmarking, debugging, tuning, and certification testing.

Finally, all of these components and tools will be unified within a framework that promotes interoperability between components, including those included in this product, those purchased from other commercial vendors, and those developed in-house by the system developer.

Infrastructure components, as are proposed here, typically do not become successful simply because they solve any single problem. Rather, they are selected for use in systems because they simplify the overall task of developing an application or a system. Thus, the ability of application developers to comprehend the components and to make effective use is critical — and is almost impossible to predict in advance. There is no substitute for exposure in real systems and feedback by real developers.

This phase II effort is intended to move a promising set of concepts from the research and non-real-time worlds and make them usable in the creation of real-time, fault-tolerant, mission-critical systems. By working with leading DoD contractors in applying these concepts to real weapons systems, we will identify ways in which to improve the utility of these components. These changes or additions can then be developed, either by The

---

[39] http://www.opengroup.org/RI/technologies/cords/

[40] http://www.cs.arizona.edu/xkernel/

Open Group during a Phase III effort or by a large computer vendor after a sale of the technology.

G. Potential Post Applications

The use of distributed systems for throughput scalability and for fault tolerance is becoming more and more popular. "Computing clusters" are offered by several computer vendors, and "web server farms" is a term being heard by ordinary users of the Internet. As these technologies migrate into mission-critical systems in the commercial world, there are several identifiable target markets that have similar requirements to the military weapons systems that we propose to investigate during Phase II. One promising area is the financial community. For example, the Securities Industries Automation Corporation (SIAC)[41], which operates computer infrastructure for the New York and American Stock Exchanges, continues to use group communication to provide geographically distributed fault tolerant operation. Automated factories are inherently distributed and the inclusion of fault-tolerant components is increasing.

The high costs for developing, maintaining and upgrading military weapons systems has been widely reported in the popular press. The difficulty of developing a coherent, reliable computing infrastructure has been echoed by DoD contractors. Both the CORDS/GIPC group communication protocol and the concept of a coherent tool kit of components are intended to directly address those problems. The utility of this concept will be proven during the Phase II effort by direct interaction with the contractors who are actually building military weapons systems for the Federal Government.

— end —

[41] http://www.siac.com/