

White Paper

A Reconfigurable Software Control Architecture for Advanced Life Support

Executive Summary

Principal Investigator: R. Peter Bonasso, NASA JSC ER2/Metrica Inc.
Contact: r.p.bonasso1@jsc.nasa.gov, 281 483 2738

Co-Investigator: Dr. Cheryl Martin, NASA JSC ER2/Metrica Inc.
Co-Investigator: Dr. David Kortenkamp, NASA JSC ER2/Metrica Inc.

Objective: The work will develop an integrated control infrastructure for advanced life support. It will allow for the integration of a variety of control components from different sources (e.g., academic, industry, NASA, etc.) and allow them to share data and control actions. It will produce well defined application programmers interfaces (APIs) for the different components of the integrated control system and allow any control component to receive and send information to other control components. It will also allow components to be reconfigured without interrupting the control process. Integrated control of a complex advanced life support system needs a coherent architecture in which all components can live and interact. This work provides such an infrastructure and gives the AEMC program an integrating framework in which control components from all funded research can interact.

Technical Approach: This work builds on seven years of research in providing integrated control to advanced life support systems at NASA JSC, starting with the Phase I test in 1995 (Lai-fook, K. M. and Ambrose, R. O. 1997), through the Phase III crewed test in 1997 (Schreckenghost et al 1998) and most recently in the 18 month integrated water recovery system test that ended several months ago (Bonasso 2001). At the core of this research is the 3T approach to intelligent control (Bonasso et al 1997). 3T separates the intelligent control problem into three interacting layers or tiers(see Figure 2 in the Detailed Project Plan). The lowest layer contains the control laws that interface to hardware and sensors. The middle layer contains standard operating procedures that pass parameters to the control laws and execute them. The top layer contains a planning system that selects amongst standard operating procedures to accomplish system goals given resource constraints. While our particular realization of this approach, known as 3T, has been successful for several disparate ALS systems, there are fundamental shortcomings in the current 3T implementation that hinder its ability to handle the hardware and software variations anticipated in the next-generation ALS systems. In particular, it is difficult to interface 3T to outside control components that may come from academia, industry or other government agencies. We propose to re-implement 3T

using new distributed processing methods and tools and we propose to create detailed interface specifications between the 3T control system and other control components. This will provide an integrating framework in which a trusted and proven ALSS control system (3T) can be augmented with new research in intelligent monitoring and control that will come out of the AEMC NRA and other NASA and non-NASA research.

Our proposed approach is to provide a common set of interfaces (APIs) that allows any combination of control modules to be interchangeable. These interfaces must be easily specified in a standard manner and must allow a straightforward incorporation of code changes on either side of a given interface. We propose using the Common Object Request Broker Architecture (CORBA) ((OMG, 2002)) to define and implement these interfaces. CORBA has been used extensively in a number of distributed processing applications, providing an open infrastructure for computer applications to work together over Ethernet networks. CORBA is independent of both operating systems and programming languages, and tools for developing interoperable CORBA applications are available from many different vendors. Recasting the 3T control paradigm in the CORBA framework will allow control components that implement the same interface to become interchangeable with a minimum of recoding, recompilation or hardware downtime.

We will reimplement all three layers of the 3T architecture using CORBA and define explicit interfaces to those three layers. At the lowest layer (containing the control laws) we will implement a CORBA/DCOM bridge to allow access to OPC-based sensors and actuators. We will also define an interface to the control laws to allow outside vendors or investigators to provide ready-made controllers for their equipment that will fit into the control architecture. In the middle layer (containing standard operating procedures) we will define CORBA interfaces for executing and changing the standard operating procedures from external control components or by the crew. At the top layer (containing the algorithms for planning and scheduling of scarce resources) we will provide CORBA interfaces that allow for updating the planning and scheduling constraints and models from outside components (such as model-based fault diagnosis systems).

As part of our technical approach, we propose to evaluate the reconfigurable control architecture in two hardware testbeds at NASA JSC: 1) The Water Research Laboratory (WRL); and 2) The Environmental System Test Stand (ESTS). In addition we have a number of subsystem and integrated simulations that will be used to perform evaluation of the reconfigurable control architecture. Extensive testing will be performed to ensure the goals of the project have been met.

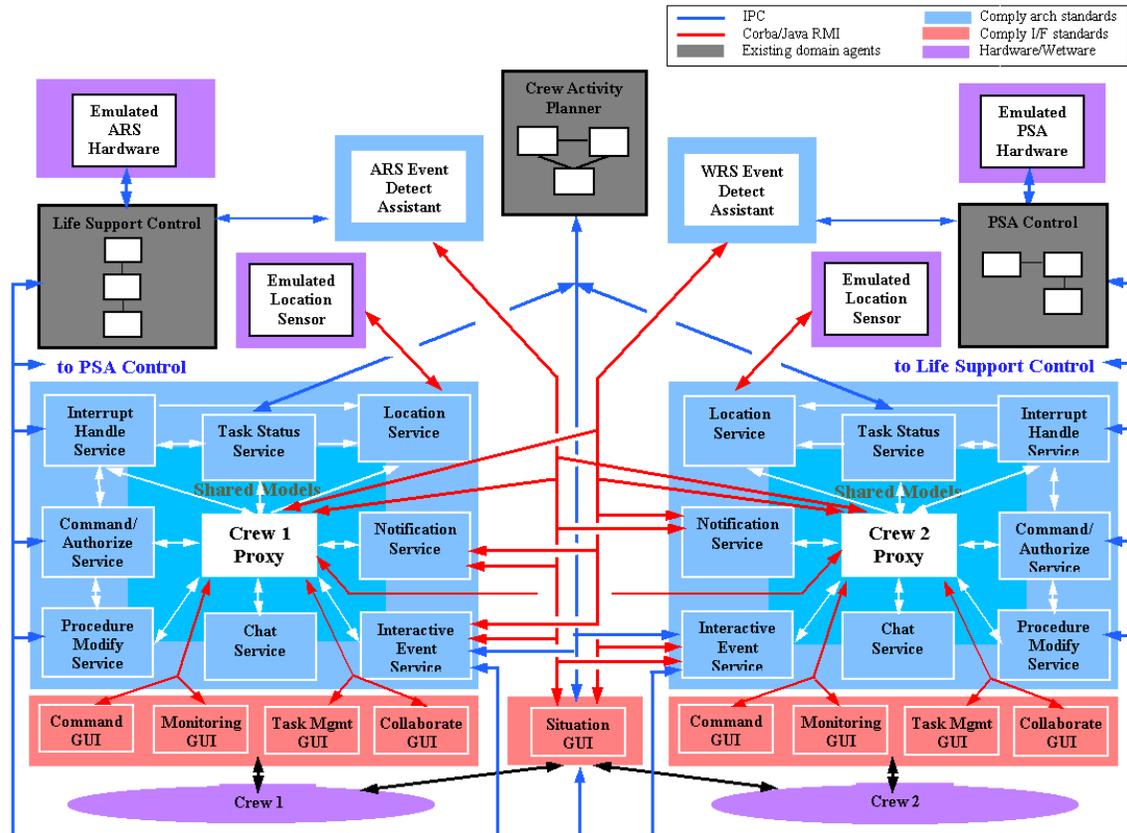


Figure 1 Crew Collaboration Architecture

The Larger Picture: Integrated control of advanced life support systems is only a small part of the larger picture, which includes crew and ground interaction with the integrated control system. Figure 1 shows a proposed architecture for crew interaction with integrated life support that is being developed at NASA JSC under a Code R research grant entitled “Distributed Crew Interaction with Advanced Life Support Control Systems” (Schreckenghost et al 2002). This work provides a crew activity planner and crew proxies that manage the interactions between crew members and the control system. The proxies are configured to display personalized control information to crew members at their workstations, PDAs or pagers. The crew collaboration architecture is completely distributed and reconfigurable. Initial testing of this crew collaboration architecture was done during the integrated water recovery system test. The work described in this proposal will create a reconfigurable control architecture that can seamlessly integrate with the crew collaboration architecture. The combined system will offer complete flexibility to both control advanced life support systems and to allow crew interaction with that control system. Efforts underway using Code R money in FY03 will extend the crew collaboration architecture to ground control. The resulting integrated system will allow everything from sensor interpretation and low-level control through executing standard operating procedures through resource and crew planning to crew and ground interaction and monitoring.

Another key component of both the reconfigurable control architecture and the crew collaboration architecture is the concept of *adjustable autonomy* (Kortenkamp, Schreckenghost and Bonasso, 2000; Dorais *et al* 1998, Barber *et al* 2001). Adjustable autonomy allows autonomous systems to operate with dynamically varying levels of independence, intelligence and control. The goal is to minimize the necessity for human interaction, but maximize the capability to interact. Our reconfigurable control architecture is designed to support multiple levels of autonomous activities ranging from direct manual control to completely autonomous control. The crew collaboration architecture is designed to support crew insight and control of the autonomous system, including adjusting its level of autonomy. The two together support a seamless control infrastructure that implements adjustable autonomy.

Resources: This project will require 27 staff months for the investigators for a total loaded labor cost of \$360,000. Additionally, we will need \$12,300 for hardware and software purchases. This totals \$372,300 for the project. We are projecting an 18-month schedule so the cost for FY03 will be \$252,300 and the cost for FY04 will be \$120,000.

Schedule: An 18-month schedule for accomplishing the our task is given in the detailed project plan. Reimplementing the three layers of 3T can be carried out in parallel since they are independent applications. Testing of the reconfigurable architecture will take place after experimentation. We have also scheduled time to document the new architecture and the experience of using CORBA for 3T.

Payoff: At the end of this project we will have a reconfigurable control architecture that can: 1) integrate a variety of control components coming from NASA, academia and industry; 2) integrate with a crew collaboration architecture and ground control tools; 3) allow for data distribution and logging for experimental analysis; and 4) provide a distributed control infrastructure for large-scale ALSS tests such as Integrity.

White Paper

A Reconfigurable Software Control Architecture for Advanced Life Support

Detailed Project Plan

Principal Investigator: R. Peter Bonasso, NASA JSC ER2/Metrica Inc.
Contact: r.p.bonasso1@jsc.nasa.gov, 281 483 2738

Co-Investigator: Dr. Cheryl Martin, NASA JSC ER2/Metrica Inc.
Co-Investigator: Dr. David Kortenkamp, NASA JSC ER2/Metrica Inc.

The objective of this proposal is to design, develop and demonstrate a reconfigurable architecture of control software for advanced life support (ALS) applications. Such an architecture will (1) support technology advancement by allowing rapid integration of newly developed control research for evaluation or comparison purposes in ALS testbeds, and (2) decrease downtime of ALS systems and improve operational efficiency by supporting control reconfiguration with a minimum of deactivation or shutdown of the systems being controlled. These capabilities are required in order to allow engineers to substitute new sensor, actuator or control algorithms into existing ground test environments, and also to allow crew to manage failure or degraded mode situations in deployed ALS systems. Using the approach described in this proposal, we will demonstrate this architecture within 18 months for \$372,300 on a simulated ALS water recovery system (WRS), a physical subsystem of the WRS, and a small ALS light stand laboratory at JSC. This effort is considered to be Technology Readiness Level 4, component validation in a laboratory environment.

General.

Future intelligent control systems for ALS must not only manage systems as disparate as food processing and air revitalization, but must also take into account the fact that these systems will be distributed geographically, executing on different software operating systems in different languages across a variety of computing hardware. Given these dimensions of distribution and the need to enable replacement or revision of any existing component for repair or testing, these systems should also be made easy to reconfigure. Recent advances in the field of distributed computing can support these requirements to an extent never before realized. Current tools for building distributed and cross-platform systems exhibit both excellent support for state-of-the-art distributed design methodology and improved standardization for interoperability across diverse components. The ease of implementation and maintenance for ALS intelligent control systems can be greatly enhanced by taking advantage of these advanced distributed computing tools now. As the first step, we propose to update the implementation for an existing intelligent control system architecture using these modern tools and methods.

Over the past seven years we have been providing intelligent control for a number of ALS applications. In these projects we have used a software control organization known as the three-

layer architecture, which has fundamental value for providing autonomous intelligent control to any ALS application. While our particular realization of this approach, known as 3T, has been successful for several disparate ALS systems, there are fundamental shortcomings in the current 3T implementation that hinder its ability to handle the hardware and software variations anticipated in the next-generation ALS systems. We therefore see a near-term need to modernize our implementation of the three-layer control architecture using the recently emerged standards and tools for interfacing distributed software and hardware within and between the architecture's layers. We propose to re-implement 3T using these new distributed processing methods and tools, thus bringing the power of new methodology for distributed systems to the three-layer control paradigm.

Background.

In support of the Crew and Thermal Systems Division (CTSD) of NASA's Johnson Space Center (JSC), we have successfully developed autonomous, intelligent control systems for ALS applications since 1995 (Bonasso, 2001; Lai-fook and Ambrose, 1997; Schreckenghost et al., 1998b). In these projects we have used the three-layer approach to organizing and developing the control software.

Three-layer Approach

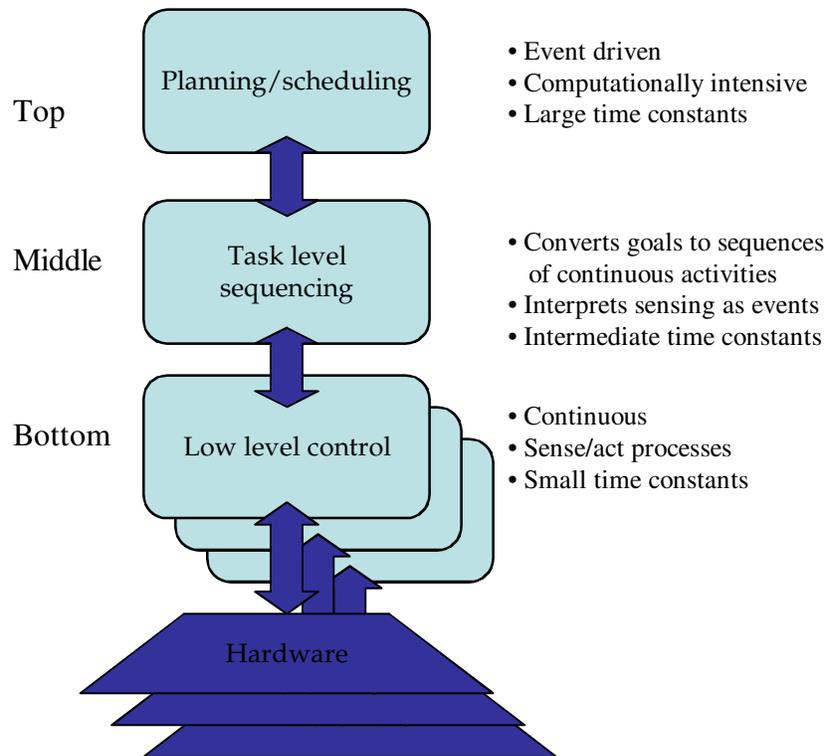


Figure 2. Three-layer Intelligent Control Architecture

The three-layer architecture, developed by the artificial intelligence (AI) community in several forms since the late eighties (a useful historical summary can be found in (Gat, 1998)), separates the general intelligent control problem into three interacting pieces (see Figure 2):

- A reactive bottom layer consisting of a set of hardware specific modules, for example, switchers, PID controllers, or data analyzers, which are tightly bound to the specific hardware and must interact with the world in real-time.
- A sequencing middle layer which can differentially activate the reactive modules in order to direct changes in the state of the world and accomplish specific tasks, for example, moving a reverse osmosis system through its phases of operation.
- A planning/scheduling top layer, which reasons in depth about goals, resources and the future effects of current actions.

The key aspect of the three-layer approach is that it gives developers the ability to integrate the continuous, near-real time control algorithms in the bottom layer with advanced AI algorithms in the top layer — i.e., automated planners and schedulers — which are event driven but more computationally expensive. The architecture achieves this integration through the action of the middle layer. Essentially, the middle layer translates the goal states computed by a planning/scheduling system into a sequence of continuous activities carried out by the lowest layer, and interprets sensor information from the bottom layer as events of interest to the top layer.

The layers allow the development and control of multiple life support equipment suites in a modular fashion in two ways. First, each layer from top to bottom can have its own data structures, timing constraints and development tools that allow for parallel development of the software. Secondly, across ALS hardware suites, the architecture allows the independent

development and testing of groups of ALS subsystems and a subsequent incremental integration of these subsystems.

3T

Since 1995, we have proven that our particular implementation, known as 3T (Bonasso et al., 1997), of the three-layer approach is a powerful organizing and development principle for the intelligent control of a contained group of life support subsystems. During a 15-day Human Rated Test (HRT), in 1995, we used 3T to monitor and "flight-follow" the control of the environment of a wheat crop chamber providing oxygen to a single crewmember. In 1997, in support of a 91-day four person HRT, we developed the software to provide 24/7 control of the transfer of product gases among an air revitalization system (ARS), a plant chamber, and a solid waste incineration system (Schreckenghost et al., 1998a) In 1999, we used the architecture to incrementally provide controls to an evolving group of water recovery subsystems, culminating in 2001 in a yearlong integrated test of a biological water processor, an inorganic removal system, a brine processing system and a post-processing system. During this test, for over 97% of the time, the control architecture operated 24/7, unattended.

Figure 3 shows the 3T model of the three-layer approach used in ALS projects to date. Our planning system develops a long-range plan for the ALS operation. Over time, it places tasks on the agenda of the sequencer for execution and monitors the results. The sequencer's plan interpreter decomposes the task into an ordered list of primitives that can be executed by the bottom layer, which is called the skills layer. A "skill" is a robust piece of code for carrying out action or for interpreting sensor information as events. In 3T we also make use of a device skill (d-skill), which manages the low-level data conversion to and from the hardware. As the action skills are enabled, event skills are activated to watch for sensor information signaling the completion (or failure) of the action. As each primitive succeeds, the old skills are deactivated and new actions and events for the next primitive are activated. When tasks complete (or fail), the sequencer informs the planner which will update the plan and send new tasks to the sequencer for execution.

While the basic idea of layering control levels based on types of activities and timing constraints remains critical to bringing advanced algorithms to bear on ALS systems, there are key shortcomings in the current 3T implementation that limit its potential for next-generation systems. In particular, the interfaces among the layers in 3T have shown they are limited in their ability to accommodate changes in hardware and software configurations, as well as to smoothly acquire data from more recent "open architecture" data systems such as Object Linking and Embedding (OLE) for Process Control (OPC) (OPC, 2002). The following section discusses the limitations of the 3T implementation in detail.

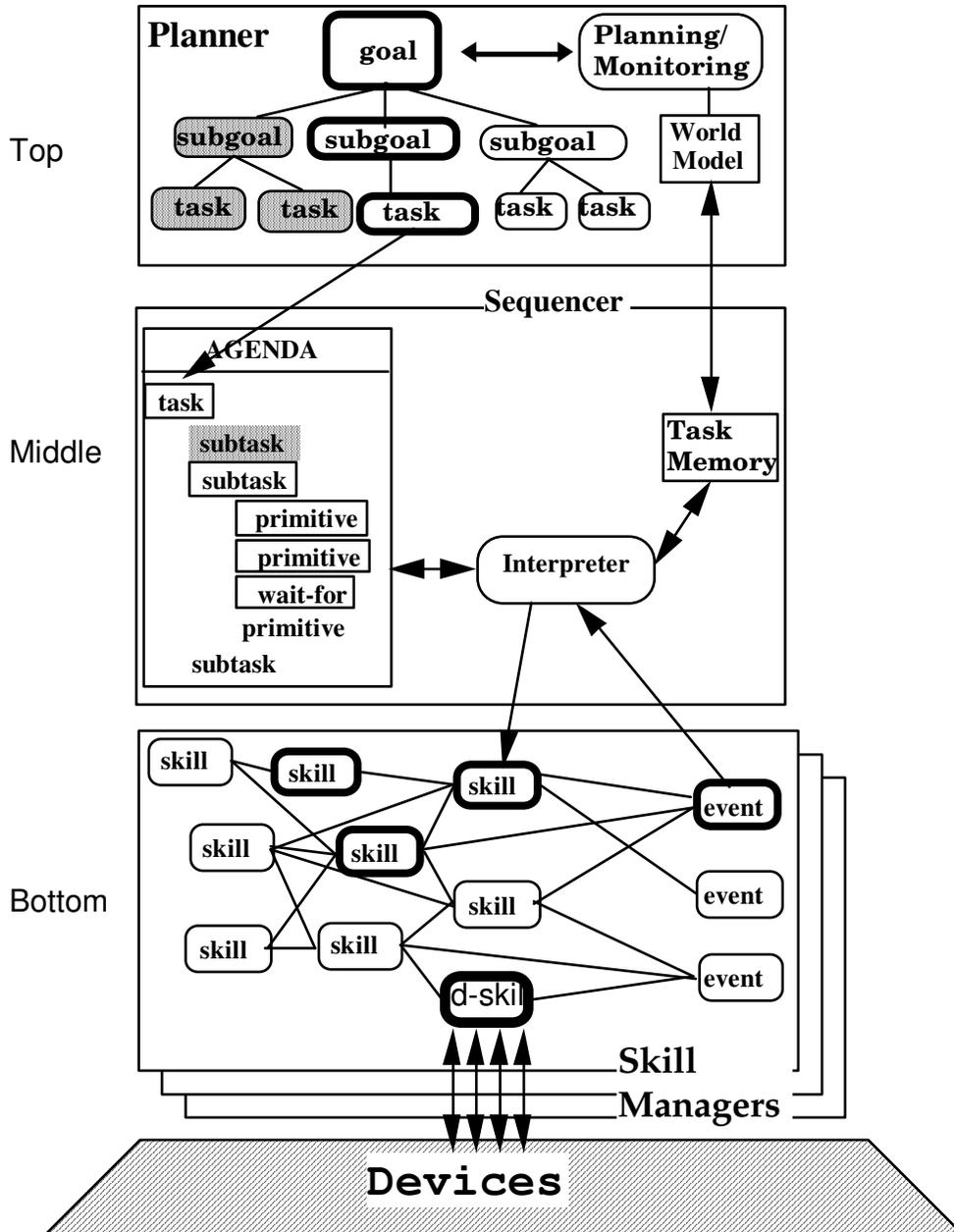


Figure 3. The 3T Realization of the Three-Layer Control Architecture

Shortcomings of the Current 3T Implementation

Figure 4, used in the following discussion, notionally depicts the 3T implementation of the three-layer architecture. The top two layers are each depicted by a single box and the bottom layer consists of small boxes representing software controlling the sensors and actuators common to a particular ALS subsystem, for example, the air evaporation subsystem (AES) and the post processor (PPS) in a water recovery system. Small horizontal bars within a layer represent the Application Programming Interface (API) from one layer to another. These APIs can be standardized for each layer-to-layer interaction, with the exception of the lowest-level interfaces to the hardware that consist of ad hoc driver software particular to the hardware being controlled.

The current implementation of 3T attempts to capture the standardization of the API between layers using static “message” definitions in a publish-subscribe message passing system known as IPC (Simmons and Dale, 1997)(NDDS is a similar type of middleware; see (RTI, 2002)). To build an interface, messages sent from one layer to another must first be defined for software on both the sending and receiving sides of the interaction. A central server receives and distributes these messages. Then code must be written to construct the messages and/or process the messages (message handlers) in any software that will send or receive them.

The boxes to the far right represent new control software that might be added to the overall architecture for testing purposes, for instance, a new scheduler (top tier), a new control algorithm (bottom tier) or new hardware. In the middle tier we might be interested in not only using a replacement sequencer, but also incorporating a new module entirely, such as a fault diagnosis subsystem that adds capability to the overall system.

There are two major limiting factors to using any publish-subscribe middleware in distributed software architectures. The first is the fact that a single server accepts and distributes the messages embodied in the interfaces, and thus can be a single point of failure in the system. The second factor is that the publish-subscribe approach has severe limitations with regard to easily adding or changing components in the architecture. The number of different interfaces, shown in the figure by the shaded horizontal strips, hints at this limitation. In effect, there is nothing

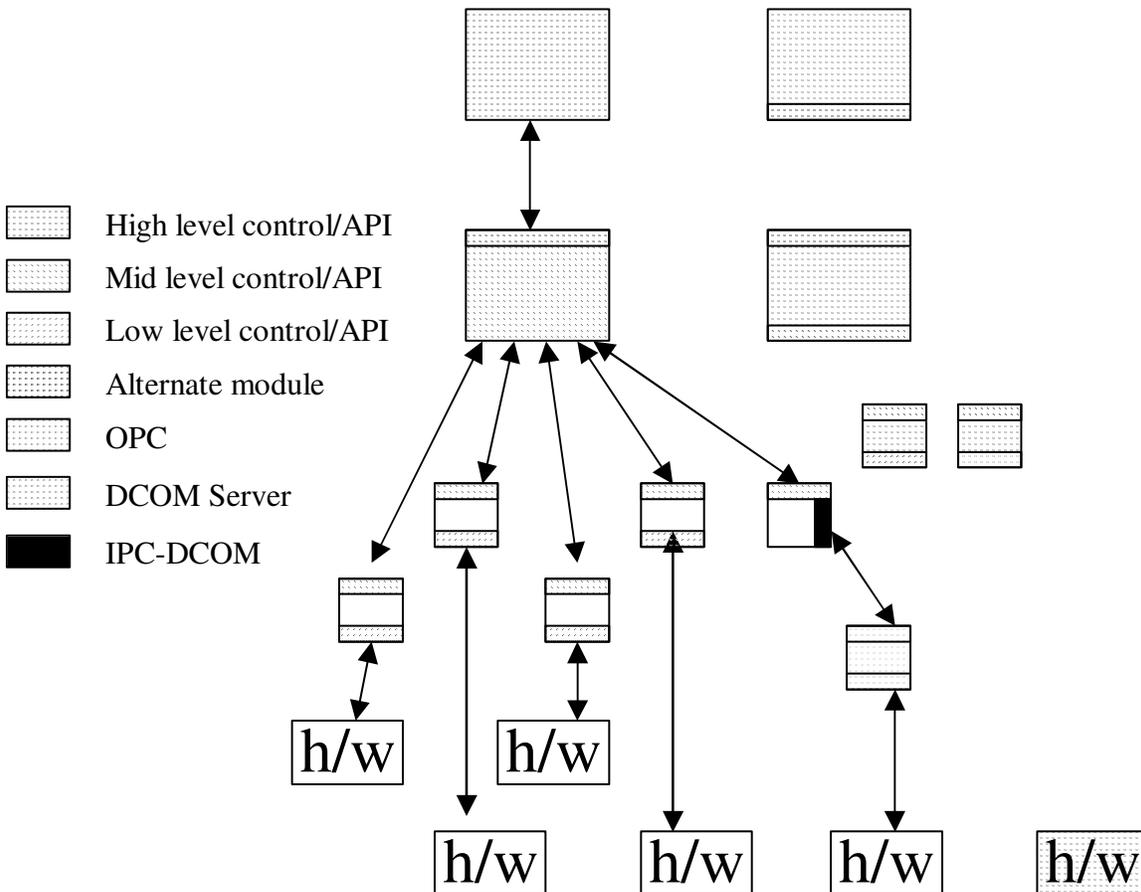


Figure 4. The various interfaces in the 3T control system. Note that the IPC-DCOM connection is notional since there is to date no implementation of such a bridge.

canonical about the interfaces, i.e., each horizontal strip represents a set of ad hoc message formats and their code handlers. There is no true standardization of the API between layers in this implementation other than the message formats themselves. The only reason that the shadings match at each level is because the programmers take care to remember (often using a text specification) which messages go with which level. So while it is relatively straightforward to add or modify a single message or handler, replacing old hardware with new or switching to a new software module at any level requires much effort not only to build but also to maintain the resulting system because message construction and processing must be coded and checked for consistency by hand.

The next two sections detail the types of changes that ALS systems will be required to accommodate in the near future, describe the difficulty with realizing those changes in the 3T implementation and discuss what types of changes are needed to create a better solution for the implementation of the three-layer architecture.

Hardware Changes.

Changes in hardware are the norm in long running, ALS environments. During deployed operations or in a ground test environment, often a sensor or actuator will fail and need to be replaced or an obsolete device will need to be replaced with a more modern version.

Affected software.

In our 3T implementation of the three-layer control architecture, the drivers for each device are compiled directly into the low-level code for that device's subsystem (e.g. the subsystem boxes in the bottom layer in Figure 4). So when one device must be replaced for that subsystem (even if the replacement device is identical as it would likely be for a repair to a deployed system), the software for the entire subsystem must be halted. This means the entire subsystem must be taken offline for the duration of the replacement procedure. Further, if the device is not identical, the device driver need to be replaced and the code for the corresponding subsystem code needs to be recompiled and tested with the replacement device (e.g., in ground test environments, the replacement device may perform the same function but in a different way such as in the case where a temperature was originally measured with a thermocouple and the replacement sensor is a digital thermometer). In some cases, a change of device will lead to additional required changes in the control code beyond the drivers (e.g., an entirely new data processing algorithm may be needed to process the new digital thermometer readings). This type of code change also requires the entire subsystem's code to be halted, recompiled, and tested. While this is obviously better than bringing the overall system down (e.g. the air evaporation subsystem as opposed to the overall water recovery system), we submit that when replacing devices, only the part of the control system affected by those devices should be disabled (at a finer resolution than the subsystem distinction).

Furthermore, even for small changes in device operating characteristics (in the bottom layer), the data the middle layer "sees" will sometimes be different for the new device. In the 3T implementation, this requires manually intensive changes to be made to IPC message formats and handlers between the bottom and middle tiers. What is required to better support the hardware and lower level software changes described above is an interface to and from the hardware to the using software that has the ability to differentially select sets of devices, and that automatically maintains consistency across the interface.

OPC Is Not Enough.

Already on the horizon there is some hope for this hardware dilemma. Manufacturers are beginning to produce sensors and actuators that feed data to new kinds of analog-to-digital (A/D) devices, which convert the data to standard formats and transmit them via Ethernet to data servers (e.g., (Iconics, 2002)) Microsoft's Component Object Model (COM) (Microsoft, 2002) specification provides the necessary software infrastructure that defines how applications share data under Microsoft operating systems such as Windows NT. The OPC specification defines a set of standard COM objects, methods, and properties that specifically address requirements for real-time factory automation and process control applications. OPC servers provide a standard interface to the OPC COM objects associated with the A/D modules, allowing OPC client applications to exchange data and control commands in a generic way. ALS laboratories at Ames Research Center and Johnson Space Center are beginning to experiment with OPC for data acquisition and management.

Sensors that take advantage of the OPC approach will always provide their data in a standard format to the rest of the control system, regardless of their operating characteristics. So in the case where a digital thermometer replaces a thermocouple, a temperature in degrees C is served by OPC from either sensor. Further, while the sensor is being changed out, only one channel from the server is affected, rather than all the channels, as is the case with hand written code at the bottom level of 3T.

However, while OPC can solve some of the problems associated with changing out hardware, it is not a complete solution. OPC is particular to the Windows operating system, and while 3T runs on any operating system, there is yet no general interface to OPC from publish-subscribe middleware. Further, many ALS devices now in use and planned for use for the next several years do not adhere to the OPC standard. So a requirement still exists to write 3T bottom layer code to interface non-OPC devices these devices to the rest of the control system, and the problems described earlier will still be present. Finally, because OPC serves only as an interface to sensors and actuators, it only partially addresses problems associated with interchanging software at the bottom layer of the three-layer architecture, and does not address similar problems in the middle and top layers as discussed in the next section.

Software changes.

In a ground test environment, particularly ones involving large numbers of ALS subsystems (e.g., the Integrity initiative (Henninger, 2001)) determining the usefulness of the control software itself is often the objective of the test. For example, a new algorithm claiming to yield better accuracy from the same water quality sensor may be tested in place of the standard algorithm running at the bottom layer of the architecture.

As a matter of fact, in each layer of the 3T model, there can be multiple types and instances of control modules appropriate to that layer. Testers may wish to experiment with a new automated planner, or investigate the added value of a fault diagnosis algorithm running in parallel with the middle layer sequencer. In the case of an alternative planner, with the current 3T implementation, new static IPC data messages and new hand-coded message handlers will need to be developed and tested. Programmers must manually ensure consistency for the newly introduced software to conform to the existing 3T messages used between the planner and the middle layer. In the case of adding a fault diagnosis algorithm, new messages and handlers will need to be built for both the sending and receiving sides of the interaction between the top and middle layers. Moreover, if the code on either side of that interaction changes, either the message formats or the handlers or

both will need changing on the other side. Since, as mentioned above, the only standardization in 3T's IPC implementation is in the message formats themselves, careful development and manual coordination with the suppliers of the new software will be required, with an inevitably long testing and debugging phase. What is required to better support these types of software changes is an interface approach that unifies the changes in both data and functions and that automatically maintains consistency across the interface.

It is important to note that the three-layer approach to intelligent control was motivated by the need to use AI techniques for the "intelligence" in intelligent control applications. As such, the top two layers tend to have AI algorithms — many written in languages such as Lisp — while the bottom layer is predominantly control code written in C, C++, or a specialized hardware programming language such as LabView (National Instruments, 2002). The point is that changing messages and handler code between layers usually involves matching up two or more languages in order to achieve the integrated whole, thus exacerbating the interface development problems. So an improved interface approach should also include cross language compatibility.

Another important aspect of the control of a large suite of ALS subsystems is the need to present the users, be they crew or ALS engineers, with a consistent view of the state of the systems. This requirement calls for a well-designed user interface (UI), which often involves graphical components (GUIs) executing on a variety of hardware platforms. In the IPC implementation of 3T we have somewhat streamlined the introduction of UIs and GUIs by developing a standard set of IPC messages for such interfaces. But again, if ground test or personnel running deployed systems wish to use different UIs or UIs designed and built by researchers at other institutions, a manual level of effort similar to that of adding software to the middle layer will be necessary to code, test and debug the required IPC formats and associated handlers for those UI software packages. Again, as in the case of software changes in the layers of 3T, what is required is an interface approach that unifies changes in both data and functions, is compatible across programming languages, and executes on the majority of hardware platforms.

In summary, to solve these software issues as well as the software problems arising from the introduction of new or different hardware, we need a more general approach to distributing the software in a multi-layered system such as 3T. Such an approach should (1) not rely on a single server, creating a single point of failure, (2) provide between any pair of software modules a canonical description for both data and functions of the interface that could generate interface code particular to the language of either module, (3) accommodate both ad hoc and OPC based sensor and actuator processing, and (4) operate on any computing platform in any geographical location reachable by high-speed networks. The next section describes a solution that meets all of these requirements.

A Better Solution.

As discussed above, we have proven the efficacy of a three-layered approach to intelligent control, as well as the usefulness of our 3T implementation to build autonomous control systems. However, as the previous sections showed, the dependence of the 3T implementation on publish-subscribe middleware and static message definitions as an approximation for a true, standardized Application Programming Interface (API) makes it difficult to extend the implementation to accommodate software and hardware changes needed in next-generation ALS applications.

A more flexible approach is called for to provide a common set of interfaces (APIs) that allows any combination of control modules to be interchangeable. These interfaces must be easily

specified in a standard manner and must allow a straightforward incorporation of code changes on either side of a given interface. We propose using the Common Object Request Broker Architecture (CORBA) ((OMG, 2002)) to define and implement these interfaces. CORBA has been used extensively in a number of distributed processing applications, providing an open infrastructure for computer applications to work together over Ethernet networks. CORBA is independent of both operating systems and programming languages, and tools for developing interoperable CORBA applications are available from many different vendors. Recasting the 3T control paradigm in the CORBA framework will allow control components that implement the same interface to become interchangeable with a minimum of recoding, recompilation or hardware downtime.

A further advantage of the CORBA implementation arises from the fact that CORBA can interoperate with the Distributed Component Object Model (DCOM) and OPC. Recall that hardware manufacturers are beginning to address some of the interchangeability and reconfiguration problems associated with maintaining systems of sensors and actuators by using DCOM/OPC to standardize data formats and provide distributed access. Through a DCOM/CORBA bridge, our proposed three-layer CORBA implementation can leverage these advances in sensor and actuator technology where they are available.

However, a logical question to ask is, “Why not use DCOM protocols instead of CORBA for all the 3T interfaces?” There are a number of reasons to choose CORBA over DCOM for applications with characteristics like the proposed three-layer control system (Geraghty et al., 1999). Our experience developing ALS systems shows that a flexible ALS control architecture must accommodate a variety of computing platforms with different operating systems including Unix, Linux, and Windows. CORBA supports all of the relevant computing platforms equally, but DCOM’s support is heavily oriented to Windows platforms (DCOM support for some versions of Linux and Unix is currently available for some versions of C and C++, e.g., (Software AG, 2002; TechnoSoftware, 2002)). In addition, many of the AI algorithms required for intelligent control in the upper layers of the proposed three-layered implementation will use Lisp as the programming language. Although CORBA supports Lisp implementation, DCOM does not. In general, CORBA exhibits greater cross-language support than DCOM (i.e. a middle layer component in Lisp and a bottom level component in C). CORBA therefore supports interoperability across all three layers, whereas DCOM’s OPC support is best suited and indeed was specifically developed for connecting the software to the hardware at the bottom layer only.

In addition to supporting multiple languages and operating systems, there are other reasons to prefer CORBA to DCOM. CORBA supports more powerful distributed exception-handling implementations and provides powerful services for finding distributed objects. CORBA is also easier to access because an application interacts with CORBA through native code generated by CORBA in that application’s programming language, whereas an application must interact with DCOM by accessing binary executables or Dynamic Link Libraries (DLLs), using various pre-defined binary offsets to call different functions. DCOM often requires designers to use direct knowledge of the interface to binary COM services and interact directly with the Windows Registry (Raj, 1998). Finally, as we will show in the next section, CORBA provides a standard solution to the question of integrating into the control system sensors and actuators that do not conform to the OPC specifications.

The investigators for this proposal have first hand experience in developing distributed CORBA applications from the ground up. One such application involved supporting a human watch team for an integrated water processing facility at NASA-JSC (Schreckenghost et al., 2002). In only a few months time we had designed, developed and connected some twenty-six CORBA objects to

the output of the water processing control system to watch for shutdown problems in the facility. These objects included a crew activity planner and task management, location and notification services for three human engineers, as well as user interfaces implemented as both graphics and notifications via paging and email. Without CORBA, such an endeavor would have taken two or three times longer to implement. Further, the complete set of object modules, written in Java and Lisp, could be executed simultaneously on different computers in multiple locations using both Windows and Linux operating systems.

The following section describes our approach to reconstructing the 3T model of the three-layer control architecture in the CORBA framework and details how the re-implemented architecture can support the hardware and software reconfigurations required by next-generation ALS systems.

Design of 3T Interfaces Using CORBA.

This section describes our approach to re-implementing the 3T interfaces using CORBA. We will describe the planned CORBA implementation at each level of 3T as well as how the interfaces will allow the integration of new or alternative code modules at that level. Prior to this description a short discussion of the Object Management Group's (OMG, the authors of the CORBA specifications) Interface Definition Language (IDL) and its associated processing is in order. OMG IDL is used to define APIs to be used by CORBA-enabled processes.

How CORBA Works.

The IDL is best described using an example. Figure 5 shows a simple IDL definition that declares the interface between the top-layer planner and any sequencer at the middle layer of the architecture. Although IDL syntax is similar to C++, an IDL file can be compiled by a CORBA tool called an "IDL compiler" into various programming languages, e.g. C, C++, Java, Lisp, Smalltalk, ADA, Python, COBOL, etc.

In the Planner module, shown in Figure 5, there is one interface that defines an object class called ForAllSequencers. The interface defines a method, called reportNewEvent, which can be invoked on an instance of that class. We can see that the method has four arguments: the name of

```
module Planner {  
  
    interface ForAllSequencers {  
  
        // A method used by any sequencer to report a new event to  
        // the 3T Planner.  
  
        void reportNewEvent( in string sequencer_name,  
                             in int event_id,  
                             in 3T::eventType event_type  
                             in 3T::eventStructure event );  
  
    };  
};
```

Figure 5. An example interface definition for a planner and any 3T sequencer

the sequencer, the event identification number and type, and the event data itself. Although the details of the data structures for the method's arguments are not shown here, they will also be completely specified in the IDL. Essentially the IDL shown in Figure 5 (the term IDL is used for both the interface definition language and a file written in the language) states that, to report an event, a sequencer must remotely invoke the reportNewEvent method with four arguments of the given type on an instance of the ForAllSequencers object in the planner module.

As shown in this example, the IDL defines an API by specifying the names and arguments of methods. For all major languages, CORBA provides an IDL compiler that automatically generates interface code, i.e., the mechanics of sending and/or receiving. Using the OMG IDL language to define APIs and an associated IDL compiler to generate interface code relieves the programmer of the tedious burden of message format consistency-checking and message handler creation and maintenance. All that remains for the programmer is to implement the "meat" of the function to perform the desired logical operations or carry out the desired algorithm. In this way, using CORBA tools makes development and maintenance of distributed systems much easier than using a publish-subscribe system with static message definitions. The following paragraphs describe the details of implementing a system with the IDL definition in Figure 5.

A CORBA IDL compiler or "stubber" processes an IDL into two sets of language-specific interface code, one set used by a process sending/making a remote function call across a network (the client) and one set used by a process receiving the remote function call (the server). These sets of generated code are then compiled or loaded into the client or server implementation. The generated code in turn interacts with a CORBA backbone implementation called an Object Request Broker (ORB) to make appropriate socket and network calls to achieve remote method invocation (RMI).

For example, if the planner were written in Lisp and the sequencer in C, the developer would execute a Lisp stubber and a C stubber on the same file to produce, respectively, code appropriate for loading into a Lisp image for the planner, and code to be compiled and linked into the sequencer C application. The generated Lisp code provides the implementation "skeleton" necessary for the planner to receive distributed calls to its executable reportNewEvent method, but the Lisp programmer is free to implement the underlying method's actions in any way to perform the associated task.

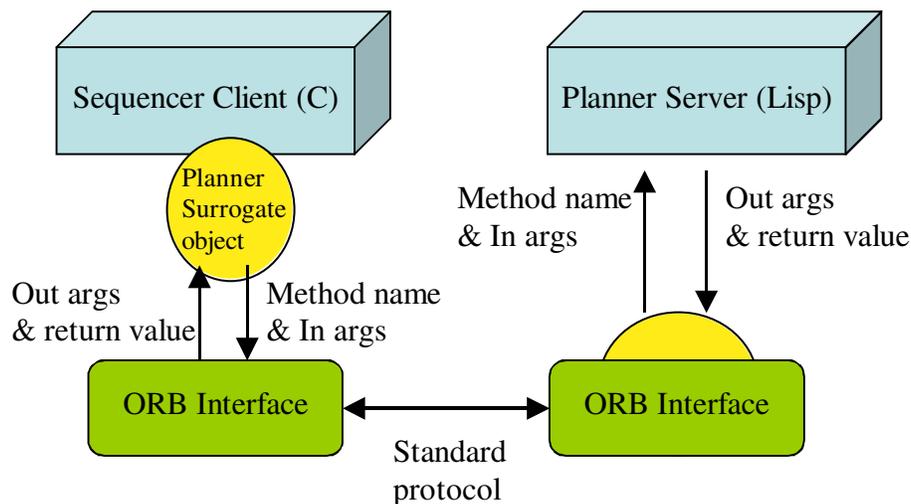


Figure 6. CORBA processing of remote method invocation (RMI).

```

module Sequencer {

interface ForPlanner {

    // A method used by any planner to post a new task
    // to the sequencer's agenda.

    void postNewTask( in int task_id,
                      in 3T::taskStructure task);

};

};

```

Figure 7. An example interface definition for a sequencer and a planner in 3T

On the other side of the interaction, the generated C code provides a “stub” that acts as a surrogate instance of the planner’s interface that receives calls from the local C code and sends them out to the remote planner object. When a new event occurs, the sequencer will call the reportNewEvent method on its surrogate object, and the CORBA-generated C code will carry out the requisite processes to communicate the method name and its filled-in arguments through the ORB to the remote planner’s generated Lisp code. The Lisp code will in turn call the appropriate method provided by the developer in the planner’s implementation (see Figure 6).

The previous example described a possible API for calls from the middle layer of the three-layer architecture to the top layer. Likewise, an IDL can be written for the sequencer to allow the planner to give the sequencer new tasks (i.e., calls from the top layer to the middle layer). In this case, the planner is the “client” making the function call, and the sequencer is the “server” receiving the call. Figure 7 shows such an IDL and Figure 8 depicts both the planner and the sequencer functioning as servers, the former for the reportNewEvent method; the latter for the postNewTask method.

In summary, CORBA gets its power and flexibility from using language-neutral IDLs as interface specifications, and by the existence of ORB implementations for all major languages. The ease

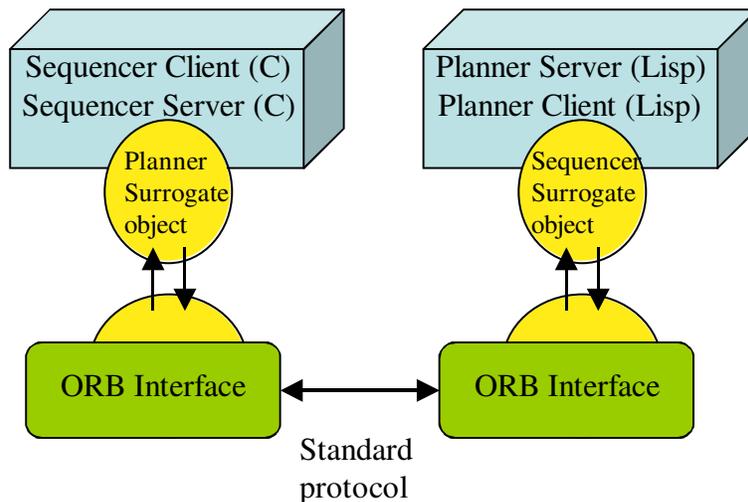


Figure 8. CORBA processing of two-way remote method invocation.

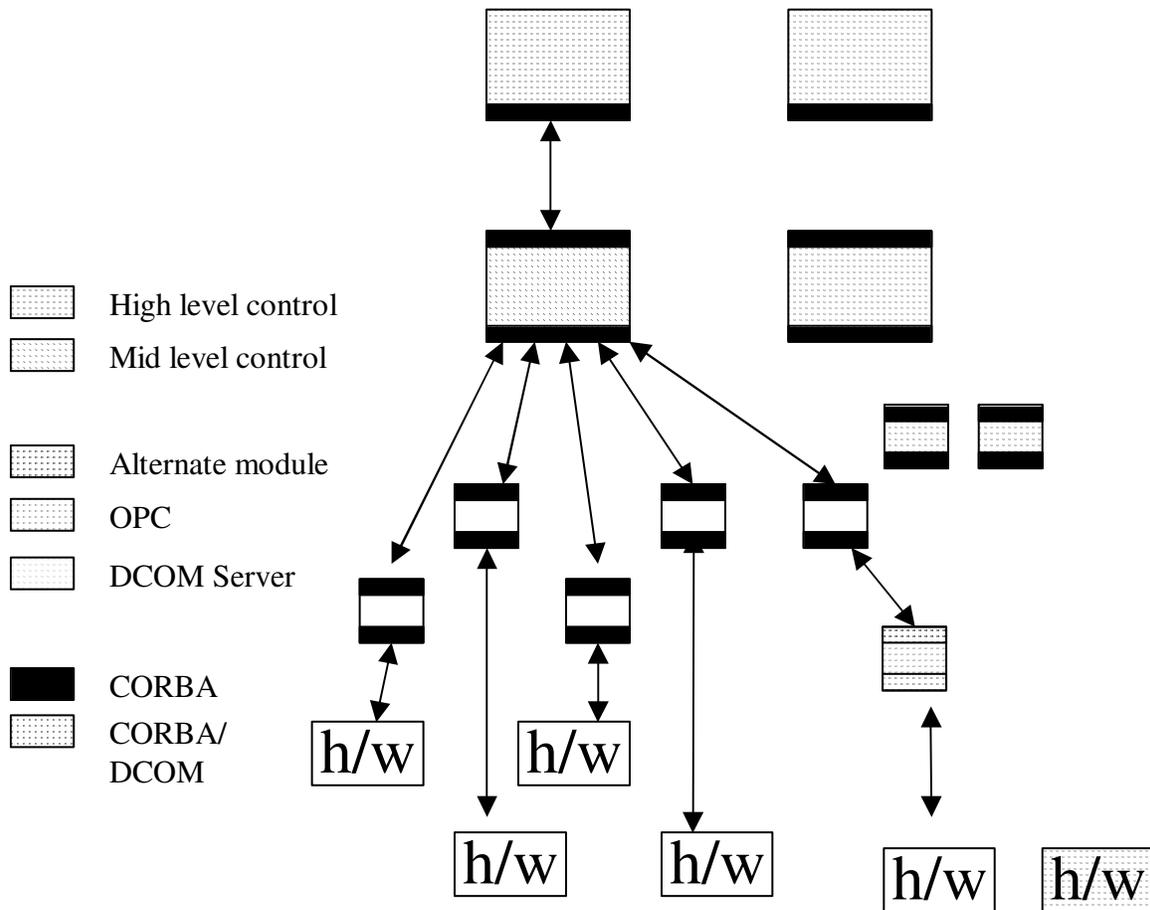


Figure 9. 3T with interfaces redesigned using CORBA

and efficiency of cross-language system integration is therefore greatly improved by using CORBA as opposed to using IPC as in the previous 3T implementation. In addition, by using IDLs and the remote procedure invocation protocol of the ORBs, CORBA avoids the two common problems of the publish-subscribe approach. First, the modules on either side of an interface can be either a client or a server, thus doing away with a single server and the single point of failure. Moreover, the IDLs serve as a standard way to specify both data and the functions associated with that data used between two modules, whereas the only standardization in a publish-subscribe approach is in the message data, for which consistency must be maintained manually across communicating processes. The implications of these differences for re-implementing 3T are discussed in the next section.

Implementing 3T in CORBA.

Figure 9 shows our view of 3T with the interfaces redesigned using CORBA. As is evident by the black interfaces in the figure, CORBA is used throughout the proposed implementation. Wherever a module cannot take direct advantage of CORBA — such as OPC servers — we will design CORBA wrappers so that those modules appear as CORBA objects to the rest of the architecture.

Implementing CORBA at the Bottom Layer of 3T

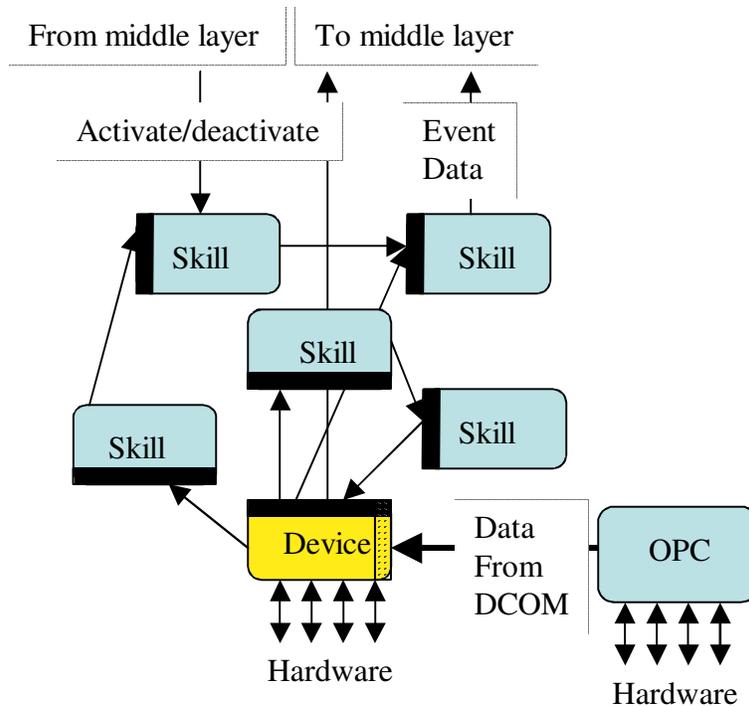


Figure 10. Notional 3T Skills Network with Device Skill

Recall that the bottom layer of 3T is called the skills layer. Skills are tight sense-act loops written in C that achieve or maintain states of the hardware to which they are connected. For each ALS subsystem being controlled there is a network of skills (see Figure 10) that can be differentially activated and deactivated by the sequencer in the middle layer. Over the years we have found it efficient to design for each skill network a device skill whose sole job is to convert the digital signals from the hardware to the data structures needed by the rest of the skills in the network. The device skill will serve as the wrapper for accommodating data from both OPC servers and from non-OPC drivers.

The CORBA implementation for the skills layer will require two steps. First, we will standardize the interface from the device skill to the other skills by developing (1) skill IDLs describing data structures and methods which the device skill can call to provide processed sensor data to the skills, and (2) a device skill IDL by which the skills can receive sensor data and command actuators through the device skill. As Figure 9 shows, it will be at the device skill that we will use a CORBA-DCOM bridge to provide access to OPC-based sensors and actuators. The Object Management Group has published a specification for such a bridge (the OMG COM/CORBA Interworking Specification) and several research groups have implemented them (Geraghty et al., 1999). As the second step, we will augment each skill IDL to include data structures and methods that can be used by other skills in the network. In this manner, the inter-skill interface definitions will be standardized as well as the interface to the hardware.

It should be noted that for some OPC-based devices, commercial COM servers provide skill-like algorithms, such as “bang-bang” controllers and sensor filters. The output of these servers will be passed directly to the middle layer by way of the device skill. Some devices use OPC servers that do not have algorithms specific to that device, such as the domain specific positioning of a multi-

way valve. In those cases, in addition to our two-step CORBA implementation, new 3T skills will have to be developed, unless such skills exist from a previous development.

Implementing CORBA in the Middle Layer

In the middle layer of 3T, we will define IDLs for the sequencer that will specify data structures and methods to be used by any skill to send event data to the sequencer. As well, an IDL for each skill will be defined that will allow the sequencer to activate and deactivate that skill (see Figure 9). Since many of the skills are invoked in a similar manner, e.g., the same skill for commanding a two-way valve may be required in several ALS subsystems, one IDL may serve several interfaces. The existing IPC message formats and associated handlers will serve as a guide for the number and types of interfaces needed in the IDLs. Therefore, instead of a large list of message formats and many pieces of handler code scattered among the layers, CORBA IDLs will organize the data and data handling in a manner not possible in the publish-subscribe framework.

As alluded to in the simple examples of CORBA IDLs discussed previously, our IDLs for the sequencer will also include interfaces for applications at the top layer of the architecture such as planners and schedulers. The sequencer not only accepts tasks for its agenda from the top layer of 3T, but also provides execution information from its memory in the form of answers to queries generated by the top layer. Methods for these data exchanges will be included in the sequencer IDL.

Using the approach outlined above, the ease of accommodating extant sequencer applications becomes clear. Alternative sequencers need only re-organize their existing code as specified in the sequencer IDL in order to communicate with the top and bottom layers of the architecture, then generate stubs from the IDL compiler in the appropriate language and compile those stubs into the new sequencer module.

For adding a new application in the middle layer, such as a fault diagnosis module, we assume the new application already has functions to accept data from a data source and to provide its processed information to a receiving application. IDLs will already exist for the skills layer, so the new application must organize the affected data and functions according to those IDLs, stub-out the IDLs and compile the resulting stubs into the application. In order for the sequencer to receive processed information from the new module, the sequencer IDL must be augmented with data structures and methods to be invoked by the new app. Then new stubs must be generated and the resulting code compiled into the sequencer application.

While the above may seem complicated, the IDLs and the automatic generation of the language specific stubs guide the process. In a publish-subscribe approach, such as IPC, new message formats must be hand-generated on both sides of the interface and then language specific handlers for those formats must be written, coded and debugged. While the logic internal to the handlers for IPC and the methods defined in the IDLs can both have bugs, the chance of bugs in the remote invocation of the functions is virtually eliminated in CORBA, whereas, the interfaces using IPC must be debugged by physically sending each message format on both sides of the interface.

Implementing CORBA in the Top Layer of 3T.

The top layer of 3T must invoke methods on the middle layer, through the sequencer IDL defined for the middle layer in the proposed implementation. To date we have not seen a requirement for applications in the middle layer to call methods in the top layer, but we can envision future needs for such methods. For instance, the fault diagnosis application mentioned in the previous section

may be able to update a planner's situation database. In such a case, the IDL for the planner would be augmented to include methods remotely invoked by the middle layer fault diagnosis application. The processing body of such methods will need to be added to the planner code, but such code would be required for any kind of implementation, and the resulting object-oriented software will be easier to modify and maintain than code resulting from ad hoc approaches.

Implementing User Interfaces for 3T

User interface code can exist at any level of the 3T architecture. Currently, a standard set of IPC messages exists to communicate skills, sequencer and planner data to the user. Using these messages and their handlers as guides, we will define IDLs for interfaces at each level of the architecture.

Multiple Modules at Each Layer.

Using CORBA, any number of control modules can be defined at each level and sets of them can be configured at the start of an operation. Key to this configuration is the fact that CORBA provides a Name Service, which serves a directory of the names and locations of CORBA objects running in a CORBA application. When each object starts up, it registers its location information with the Name Service, thus allowing other objects to locate and use the services of that object. A control system startup script can define the number and types of CORBA objects to be used in any given ground test or deployed configuration of the control system. Thus, only those objects designated to "run" will register with the Name Service. In this manner, ground test engineers can easily switch out alternative control algorithms at any level. For instance, if two sequencers have been developed, the start up script may designate the second one to be active, and the planner and the skills can lookup the designated sequencer once it registers with the Name Service. Alternatively, we can make use of CORBA's Trading Object Service, which allows modules to find other modules based on their capabilities, rather than their names. This "matchmaking" service allows each object to register a description of its capabilities, which in turn allows other objects to find that object based on this description (i.e., a yellow pages rather than a white pages).

This flexibility cannot be achieved with publish-subscribe middleware. Although, with a single server, there is no need to lookup other modules, such middleware does not provide any utilities to determine at startup which modules are available with which message formats. For the publish-subscribe system, such utilities must be built in an ad hoc manner by the applications developers.

Benefits of a CORBA-based 3T Control Architecture.

As shown in Figure 9, 3T will define all of its interfaces using CORBA IDLs. When we complete this redesign, 3T will be a true plug-and-play control architecture, allowing the change out of sensors and actuators, and control and interface modules at any level of the architecture. In addition, a ground test team or deployed crew will be able to specify at startup, which modules are to be active, and these modules will automatically find each other and configure themselves appropriately.

The new CORBA-based 3T will be able to accommodate all the control changes discussed in earlier sections, in particular:

- 1) Device failure. With a separate interface for each device defined in the low-level IDLs, the using modules can determine if that device is out of commission through the exception handling facilities that CORBA provides, and they can avoid invoking the interface code for that device. Code for operational devices will be unaffected.
- 2) New device. Once a replacement has been made, if the control algorithm does not already exist for the device, e.g., in the existing skills or in the appropriate OPC server, new methods may need to be defined in the IDL for the middle layer. Otherwise, a new control algorithm will be required, but with the same middle layer interface as the existing skills.
- 3) New low level control algorithm. Since the device skill and middle layer interfaces are already defined, it is only necessary to stub the IDLs in the language of the new algorithm, match the stubbed code to the functionality of the new algorithm, and then compile them together into the code for the new skill. At startup, the new skill will be made active and the other skills will locate it via the Name Service.
- 4) Using a different sequencer. As in the case of a new control algorithm, since the interfaces for the top and bottom layers are already defined, it is only necessary to develop the new sequencer's interface in accordance with the existing IDL definitions.
- 5) Using a new middle layer application. The new application must organize its affected data and functions according to the existing bottom-layer IDLs (sequencer/skills) and top-layer IDLs (sequencer/planner). If the application needs to invoke new methods in any of the layers, those IDLs must be augmented with data structures and methods to be invoked by the new application, then stubbed, implemented, and compiled into the using applications. At startup, the new application will be made active and the control code at the requisite layers will locate it via the Name Service.
- 6) Using a different planner/scheduler. Developers must stub out the sequencer IDL in the language of the new or legacy planner/scheduler, match the stubbed code to the functionality of the new planner/scheduler, and compile the resulting code into the planner/scheduler application.
- 7) New user interface. All that is required to attach a new user interface to any layer is to develop that user interface in accordance with the UI IDL defined for the appropriate layer. At startup, the new user interface will register to receive data from the control software for display.

In addition to the above capabilities, it will be possible to host any control software at the appropriate level in the architecture, and to execute that software on any combination of computing operating systems at any combination of locations reachable by Ethernet.

Plan for Accomplishing the Work.

To realize the design described in the previous section, we will take a three-phased approach, suggested by the three layers of the architecture, i.e., use CORBA to re-implement the skill layer, the sequencing layer and the planning layer. In addition, we will develop a configuration methodology to demonstrate customizing the number and type of control modules used in a given run of the control system.

The application backdrop will be the control of an integrated water recovery system (iWRS), previously developed and run autonomously for two years (Bonasso, 2001). The iWRS consists of four subsystems: a biological water processor (BWP) for the degradation of total organic carbons and the removal of ammonia, a reverse osmosis (RO) system for the removal of inorganic salts, an air evaporation system (AES) to recover water from the brine produced in the RO, and a

post processing system (PPS) to remove trace organics and inorganics, resulting in potable water. Each subsystem has its own skills layer; a single sequencer orchestrates all four skill sets. The top layer consists of a scheduler used to arrange maintenance operations for the iWRS.

The location of the work will be the Intelligent Systems Laboratory (ISL), the Water Research Laboratory (WRL) and the Environmental System Test Stand (ESTS) at JSC (see Laboratories for Carrying Out the Work).

Phase 1: Skills Layer

Implementing CORBA for the skills layer consists of three efforts: re-implementing the existing skills in CORBA, re-implementing the skill level user interface, and incorporating the CORBA to DCOM bridge for OPC-based devices.

Re-implementing CORBA in 3T skills

Using the existing skill sets from the iWRS, we will design IDLs for the devices skills and the intra-skill interfaces, generate the stubs, incorporate new CORBA methods, and test the resulting skills against a skill level simulation previously developed for the iWRS test. After the basic implementation has been tested in the ISL, we will have an opportunity to test the connection between actual devices and the device skill on the PPS, which has been left intact in the WRL for further subsystem testing (see Laboratories for Carrying Out the Work).

Re-implementing the Skills User Interface

For the iWRS test we developed graphical user interfaces (GUIs) that display data broadcast by the device skills. To realize this connection in CORBA, an IDL for the GUI will be defined and used by the device skills to post new signal data at regular intervals. Aside from adding CORBA specific code to receive the data, the code for the GUIs will remain unchanged.

Building the CORBA to DCOM Bridge

Based on the OMG COM/CORBA Interworking Specification, a number of commercial vendors have developed bridging tools to support interoperability between DCOM and CORBA (Geraghty et al., 1999). In this effort we will survey the set of existing CORBA to DCOM bridge implementations and select one among them for the basis of our bridge development. We expect the majority of this effort will be spent incorporating the bridging software into our ISL computing and network infrastructure. After the basic implementation has been tested in the ISL, we will test the bridge with a suite of OPC-based hardware controlling a plant lighting stand in the Environmental System Test Stand (ESTS, see Laboratories for Carrying Out the Work).

Phase 2: Sequencer Layer

Using the existing 3T sequencer software from the iWRS, we will (1) recast the existing sequencer activation, deactivation, query and event functions used in the skill level as methods defined in the skill IDLs, and (2) define the sequencer IDL and build the methods used by the skills to communicate data to the sequencer. With the sequencer written in Lisp and the skills in C, this effort will demonstrate the cross language remote method invocation capability of CORBA.

Additionally, the methods used by the sequencer to display data and take commands from the sequencer GUI will be re-implemented as CORBA methods defined in a GUI interface to the sequencer IDL. The result will be an example of separate threads in the same language process communicating via the CORBA specification.

Phase 3: The Top Layer

Currently, the iWRS maintenance scheduler runs in the same thread as the sequencer. For this phase of the CORBA development we will (1) extend the sequencer IDL to include an interface for the scheduler, (2) define a scheduler IDL with an interface for the sequencer, and (3) implement the methods and run the scheduler on a computer other than the one used by the sequencer. The separation of scheduler and sequencer serves as a precursor to the reconfiguration demonstration described next.

Reconfiguring the Control Architecture

In order to demonstrate reconfiguring the control architecture, we will make duplicate copies of selected skills, the sequencer and the scheduler. We will define a configuration script that will designate which of the skills, sequencers and schedulers will be used at the start up of the control system and demonstrate changing modules in the architecture by simply making changes to that script.

In addition, we will run multiple GUIs for the skills on a number of different computers, demonstrating that all the GUIs will be updated with the same data.

Laboratories for Carrying out the Work.

We will develop and test the IDLs and associated methods and data structures in the Intelligent Systems Laboratory (ISL) at JSC, and we will carry out hardware testing in two JSC ALS laboratories.

Intelligent Systems Laboratory: The Automation, Robotics and Simulation Division (ARSD) of NASA JSC maintains an Intelligent Systems Laboratory (ISL), which will provide computational infrastructure for this project. The ISL contains several dozen networked Unix and Linux workstations that are administered by a full-time support staff. The ISL also contains Windows and Macintosh PCs as well as special purpose computers, such as those running real-time operating systems for control research. Standard software packages are maintained and routinely upgraded. Software development, testing, and integration of software components will take place in the ISL.

Water Research Laboratory (WRL). The Crew and Thermal Systems Division (CTSD) of NASA JSC maintains the Water Research Laboratory, which will provide hardware to test the CORBA-based 3T architecture. The PPS of the 2000-2002 iWRS test remains operational along with the controls rack and computers used by 3T to control the iWRS systems. In order to process water in a stand-alone manner, the PPS has its own feed tank and pump. The computers are linked together via LAN; a separate computer provides a gateway to other JSC laboratories, including the ISL.



Figure 11. A Rack of iWRS Subsystems in the Water Research Laboratory

Environmental System Test Stand (ESTS). The Environmental System Test Stand (ESTS) at Johnson Space Center is used as a testbed to prototype various subsystems for the plant growth chamber, called the BPC (Biomass Production Chamber). The ESTS is being used for testing the operating characteristics of different lighting systems and for testing control system hardware and software components. The control system hardware is currently comprised of National Instruments Fieldpoint I/O devices and Sixnet Ethertrak I/O devices connected via Ethernet. Each of these devices has an OPC server running on computers connected to the same Ethernet that reads data from the devices and has the ability to distribute it to any OPC client on the network. The server for the Windows NT network is a 700 MHz Pentium III computer with 768Mbytes of RAM, running Windows NT 4.0 Server edition. There is also a DIN rail mounted PC from SBS Technologies being used on the network to forward the Fieldpoint and Sixnet data over the network. It is a Pentium III 700 MHz machine with 256 Mbytes RAM running Windows NT 4.0 Workstation edition. A commercial off the shelf software package called Object Automation from Wellspring Solutions is used for data acquisition and control. This package has built-in alarming and redundant data storage capability. It also serves as both an OPC client and server. Object Automation allows for methods to be written in Java, IEC 61131-3 compliant Ladder Logic, or IEC 61131-3 compliant Functional Block diagrams. All components of the application can be distributed across the NT network.

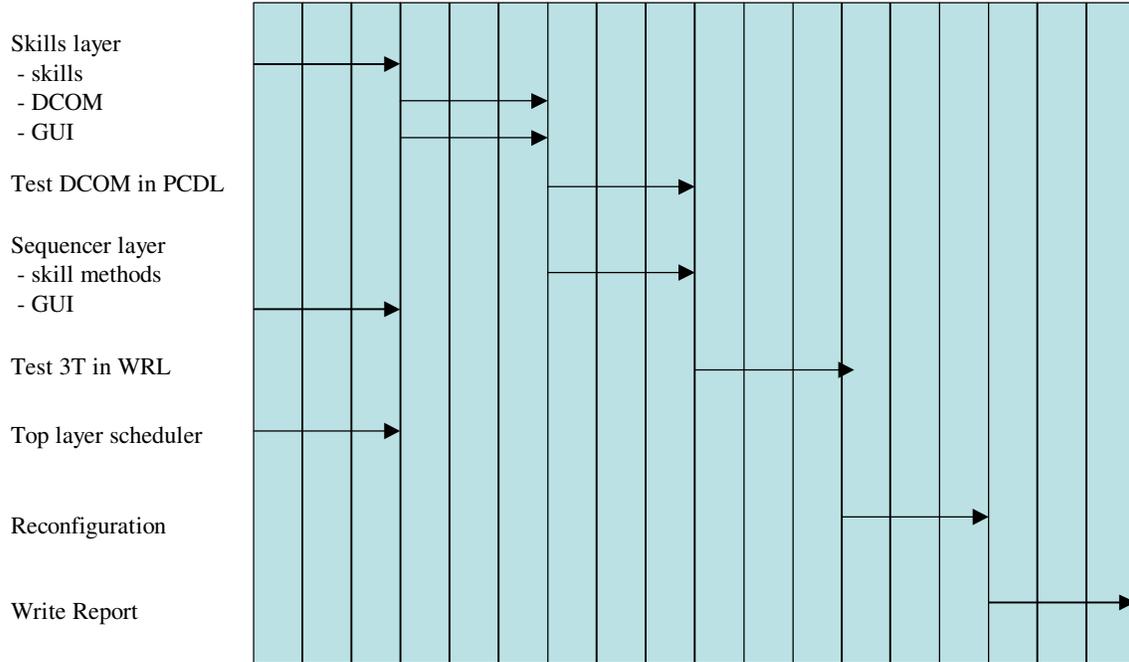


Figure 12. Environmental System Test Stand

Schedule

An 18-month schedule for accomplishing the above plan is shown below. The sequencer tasks not involving the skills layer as well as the scheduler tasks can be carried out in parallel since they are independent middle and top layer applications that do not require having CORBA implemented in the bottom layer of 3T. We have included a report task to document the experience of using CORBA for 3T and comparing its operation with that of the IPC-based version used for the iWRS test.

01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18



Costs

We estimate the total labor costs for the above schedule to be 27 staff months or \$360000, using the contractor labor rates for TRACLabs personnel.

Additionally, we will need \$12,300 for hardware and software purchases, including:

- Orbix 3, including Iona's DCOM/CORBA "COMet" bridging tools, (\$7300),
- Windows 2000 software development environment for C++ and Java (\$500),
- and a Windows/Linux laptop to be used in the ESTS for testing the DCOM-CORBA bridge (\$4500).

The remainder of the software required for this project (i.e., the Lisp, Java, and C++ programming languages and development environments for Unix/Linux, 3T development tools, OPC development environments and servers, ILU ORB for CORBA/Lisp development, JacORB for CORBA/Java development, and TAO for CORBA/C++ development) already exists in the ISL and ESTS facilities.

Personnel

Investigators at JSC and at the TRACLabs division of Metrica, Inc., both located in Houston, TX, will carry out the scheduled work for this project. As principle investigator, Pete Bonasso will manage the project execution as well as carry out the implementation of CORBA for the top two layers of the architecture. Mr. Bonasso is a principal designer of the 3T intelligent control architecture, and since 1995, has investigated the application of 3T to advanced life support systems, where the key requirement has been the minimization of human vigilance. He has applied 3T to the monitoring and control of plant nutrient delivery systems and an advanced air revitalization system (ARS), and he developed the system planner for managing crew/plant gas exchange during a ninety-one day ALS human test. Since 1998, he has been the designer and developer of the 3T control system supporting the two-year iWRS test. Mr. Bonasso is a co-founder of the Annual AAAI Robot Competition and Exhibition (Dean & Bonasso 93, Bonasso & Dean 97). He has served on the program committees for the National AI Conference and the annual Agent Theory, Architectures, and Languages conference. He is also an editor of *Artificial Intelligence and Mobile Robots*, with David Kortenkamp and Robin Murphy, published by the MIT Press in 1998.

Dr. Cheryl Martin will lead the overall design for integrating CORBA into 3T, in particular, the development of the IDLs for each level of the architecture. She will also design and guide the development of the CORBA to DCOM bridge for this application. Over the past four years, Dr. Martin has acted as a principal designer for two other complex distributed systems based on CORBA: the Sensible Agent Testbed at The University of Texas (Barber and Martin, 2001; Barber et al., 2000) and the Distributed Crew Interaction project at JSC/TRACLabs ((Schreckenghost et al., 2002)). In conjunction with these projects, Dr. Martin has published several research papers discussing design and required infrastructure for distributed systems supporting AI technologies (Barber et al., 2001; Barber et al., 2000). She has expertise in code development on both Linux and Windows platforms in a variety of programming languages including Java and C++. Dr. Martin also has an electrical engineering background that includes design and development for several real-time control and data-acquisition systems at NASA LaRC for both wind-tunnel instrumentation and robotics applications.

Dr. David Kortenkamp has worked at TRACLabs in support of JSC intelligent robotics programs since 1994. A co-designer of the 3T intelligent control architecture, he will lead the design and development of the CORBA related code changes to the skill managers. Dr. Kortenkamp has worked on a number of ALS control applications, including the 15-day early human test, the Node 3 advanced ALS technology demonstration, the 450-day biological water processing test, and the iWRS test. Dr. Kortenkamp was chair of the 1999 IJCAI Workshop on Adjustable Autonomy Systems, and served on the program committee for the National Conference on Artificial Intelligence and as a guest editor of a special issue of the Journal of Experimental and Theoretical Artificial Intelligence devoted to intelligent control architectures. He is also an editor of *Artificial Intelligence and Mobile Robots*, with Pete Bonasso and Robin Murphy, published by the MIT Press in 1998.

Metrica/SKE personnel at JSC will perform software implementation and integration.

References

- Barber, K. S., Goel, A., Han, D. C., Kim, J., Lam, D. N., Liu, T. H., MacMahon, M. T., McKay, R. M., and Martin, C. E. 2001. Infrastructure for Design, Deployment and Experimentation of Distributed Agent-based Systems: The Requirements, the Technologies, and an Example. In *Proceedings of the Autonomous Agents and Multi-Agent Systems (AAMAS-2001)*, 92-99.

- Barber, K. S. and Martin, C. E. 2001. Dynamic Adaptive Autonomy in Multi-Agent Systems: Representation and Justification. *International Journal of Pattern Recognition and Artificial Intelligence* 15(3): 405-433.
- Barber, K. S., McKay, R. M., Goel, A., Han, D. C., Kim, J., Liu, T.-H., and Martin, C. E. 2000. Sensible Agents: The Distributed Architecture and Testbed. *IEICE/IEEE Joint Special Issue on Autonomous Decentralized Systems* E83-B(5): 951-960.
- Bonasso, R. P. 2001. Intelligent Control of a NASA Advanced Water Recovery System. In *Proceedings of the The 6th International Symposium on Artificial Intelligence, Robotics and Automation in Space: A New Space Odyssey*. Montreal.
- Bonasso, R. P., Firby, J. R., Gat, E., Kortenkamp, D., Miller, D. P., and Slack, M. G. 1997. Experiences with an Architecture for Intelligent, Reactive Agents. *Journal of Experimental and Theoretical Artificial Intelligence* 9: 237-256.
- Dorais, G., R. P. Bonasso, D. Kortenkamp, B. Pell and D. Schreckenghost, "Adjustable Autonomy for Human-Centered Autonomous Systems on Mars," *Mars Society Conference*, 1998.
- Gat, E. 1998. Three-Layer Architectures. In *Mobile Robots and Artificial Intelligence*, Kortenkamp, D., Bonasso, R. P., and Murphy, R., Eds.: AAAI Press.
- Geraghty, R., Joyce, S., Moriarty, T., and Noone, G. 1999. *COM-CORBA Interoperability*. Upper Saddle River, NJ: Prentice Hall PTR.
- Henninger, D. 2001. A Vision. In *Briefing on the integrated human exploration mission simulation facility (Integrity)*. Houston: EC, NASA-JSC.
- Iconics, I. 2002. Web-enabled OPC Automation at Your Fingertips. <<http://www.iconics.com/>>.
- Kortenkamp, D., D. Schreckenghost and R. P. Bonasso, "Adjustable Control Autonomy for Manned Space Flight," *IEEE Aerospace Conference*, 2000.
- Lai-fook, K. M. and Ambrose, R. O. 1997. Automation of Bioregenerative Habitats for Space Environments. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 2471-2476. Albuquerque, NM: IEEE.
- Microsoft. 2002. COM: Delivering on the Promises of Component Technology. <<http://www.microsoft.com/com/default.asp>>.
- National Instruments. 2002. NI Labview. <<http://sine.ni.com/apps/we/nioc.vp?cid=1381&lang=US>>.
- OMG. 2002. CORBA Basics. <<http://www.omg.org/gettingstarted/corbafaq.htm>>.
- OPC, F. 2002. OPC Foundation Home Page. <<http://www.opcfoundation.org/>>.
- Raj, G. S. 1998. A Detailed Comparison of CORBA, DCOM, and Java/RMI. <<http://www.execpc.com/~gopalan/misc/compare.html>>.
- RTI. 2002. Introduction to NDDS. <<http://www.rti.com/products/ndds/literature.html>>.
- Schreckenghost, D., Edeen, M., Bonasso, R. P., and Erickson, J. 1998a. Integrated Control of Product Gas Transfer for Air Revitalization. In *Proceedings of the 28th International Conference on Environmental Systems*. Danvers, MA.
- Schreckenghost, D., Martin, C. E., Bonasso, R. P., Kortenkamp, D., Milam, T., and Thronesbery, C. 2002. Supporting Group Interaction Among Humans and Autonomous Agents. In *Proceedings of the AAAI-02 Workshop on Autonomy, Delegation, and Control: From Inter-agent to Groups*. Edmonton, Canada.
- Schreckenghost, D., Ryan, D., Thronesbery, C., Bonasso, R. P., and Poirot, D. 1998b. Intelligent Control of Life Support Systems for Space Habitats. In *Proceedings of the Tenth Conference on Innovative Applications of Artificial Intelligence*, 1140-1145. Madison, WI: AAAI Press / The MIT Press.
- Simmons, R. and Dale, J. 1997. *Inter-Process Communication: A Reference Manual. IPC Version 6.0*: CMU Robotics Institute.
- Software AG. 2002. *EntireX DCOM under Linux: Installation and Operation*, Technical Manual. <http://www.softwareag.com/entirex/download/dcomlinux_611.pdf>.

TechnoSoftware. 2002. OPC Solutions.

<http://www.technosoftware.com/opcsolutions/pricingusa.html#OPCLinux>.