Infotech@Aerospace
26 - 29 September 2005, Arlington, Virginia

AIAA 2005-7143

A Modern CORBA-Based Approach to Ad Hoc Distributed Process Orchestrations Applied to MDO

Robert R. Ratcliff*
FutureTek Net Services, Austin, Texas, 78726

and

Stephen T. LeDoux†, William W. Herling‡
The Boeing Company, Seattle, Washington, 98124-2207

A Multidisciplinary Design Optimization (MDOPT) framework was developed for air vehicle design and analysis. The developed MDOPT system contains a collection of technology modules for performing optimization studies that can be configured and controlled by a Graphical User Interface (GUI). A CORBA based command and control layer called the Inter-domain Communication Facility (ICF) was developed for this framework to allow the user to interactively create ad hoc workflows between remotely distributed engineering processes running interactively or within batch queuing systems such as PBS. An overview of the design of the ICF and related software technologies are presented. The interaction of the ICF process flows with the MySQL-based Database Management Facility (DMF) and the practical design considerations related to the DMF are also discussed. Finally, a sample optimization workflow that allows human interaction is shown to demonstrate the flexibility of the system.

I. Introduction

THIS paper is intended to give a review of the MDOPT[1] system with emphasis given to the architecture design and of the ICF and DMF, funded under Air Force Contract F33615-98-2-3014, in addition to Boeing cost match funds. The initial period of development was established with contract activity beginning in September 1998 and ending April 2002. Development of MDOPT has continued since then with Boeing internal funding.

The main objective for the MDOPT system was to provide a MDO framework that enabled significant reductions in design cycle times and costs, as well as, significant improvements in design quality for complex air vehicle configurations. The aim of development was to produce a solution that would gain acceptance and provide real-world usefulness, supporting tradeoff analyses to balance the demands of product performance and product cost as an objective function, without compromising flight safety. There were several system design goals that defined the operational characteristics of MDOPT to support these goals. A modular and open computing architecture was, in particular, crucial for the acceptance of MDOPT by a heterogeneous community of users. This design goal promotes the attainment of several key system characteristics: scalability — to enable the user to frame application of the system to fit available schedule requirements and computing resource availabilities for the optimization problem; flexibility — to enable the user to choose from a variety of solvers and other computer aided engineering tools; and extensibility — to enable the system to grow through refinement of existing capabilities and the incorporation of new ones (e.g., the incremental incorporation of additional analysis disciplines or enhanced geometric complexity.)

MDOPT can be described from two viewpoints, the system architecture described in Figure 1 and the system process described in Figure 2 below. The system is comprised of the six main modules as shown, where the executive or MDO Manager (MDOM) controls the overall processes and provides common utilities necessary for

* Senior Specialist Engineer, Member AIAA
† Senior Specialist Engineer, Member AIAA
‡ Associate Technical Fellow, Senior Member AIAA

general system functionality. The user interacts primarily with the system through the Graphical User Interface (GUI), but command line interfaces are also available to access most of the MDOPT functionality.

Technology discipline domains provide analysis services of candidate configuration geometries. The current MDOPT implementation provides support for three technology domains: aerodynamics, structures and stability and control (S&C). The Inter-domain Communication Facility (ICF) provides the underlying process control. It is a CORBA based communication layer composed of clients and servers written mainly in object-oriented incrTcl[2] scripts leverage the Combat[3] Tcl to CORBA[4] bindings and layered upon the MICO[5] CORBA ORB library and standard CORBA services. Transfer of user input information is achieved through a master control document that uses a namelist-style (i.e. name=list of values) format. This document represents the canonical format of the required inputs. A common parser is used to extract the required parameters from the document and various converters are available to automatically create the input decks for the various integrated processes. The master control document also acts as the conduit between the GUI, the ICF, the Database Management Facility (DMF) and the discipline domain control scripts. All persistent user data, i.e. data created by the system, is stored into a MDOPT database via DMF utilities and MYSQL[6], with the exception of large binary computational model files (e.g. flow solver restart and grid files), which are maintained separately within the system directory structure.

During interactive execution, the user inputs the required data via the GUI that updates the master control document accordingly and directs the conduct of the optimization through the ICF. At several points within the optimization process the user has access to the database and may input data manually for given discipline data or for a particular design perturbation analysis as represented by the man in the loop figures.

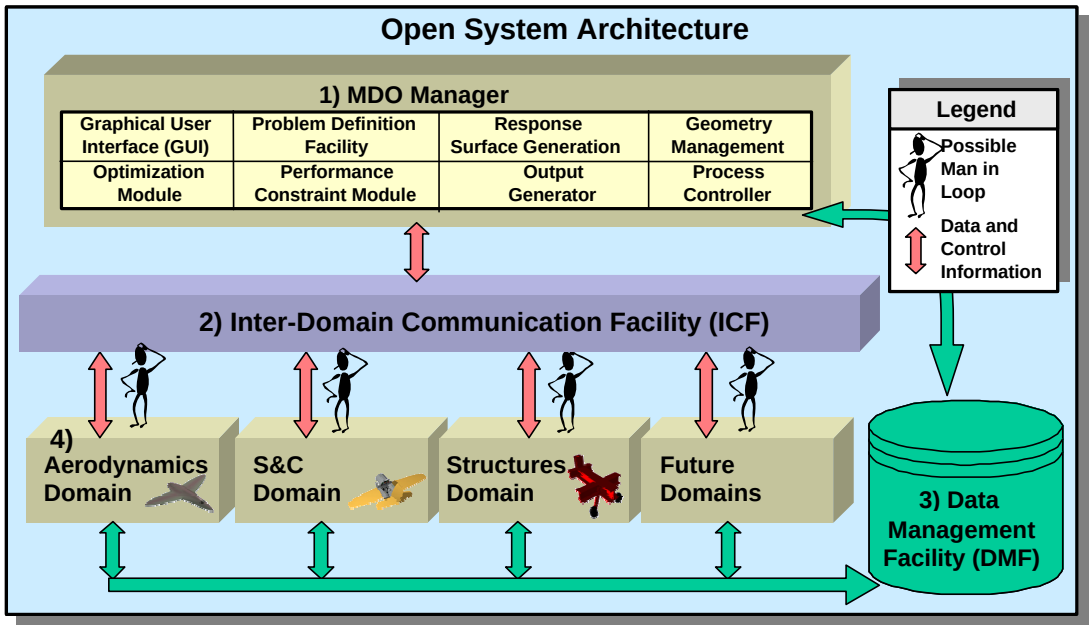

Figure 1. Diagram of the System Architecture.

The top-level components of the MDOPT system and optimization steps are shown in Figure 2. This illustration provides a general outline of the steps required in performing an optimization with MDOPT. Following a clockwise direction from starting from the top left corner, first the geometry is input in to the system and surface grids or lofts are created for the input geometry. Next the design variables are defined and a design of experiments (DOE) experiment is selected. Geometry perturbations are created for each design point in the experiment and analyzed with each of the discipline analysis codes. Geometric constraint checks are performed and stored into the database. Next Interpolated response surfaces (IRS), or surrogates, are generated for the constraints and objective functions. Optimization is then performed on the surrogates to obtain the final optimum geometry and design variable vector.

Once initial user inputs have been completed the system can proceed in an automated fashion. Parallelization of the optimization process has been implemented within most domains providing a capability to utilize large scale, multiple CPU computers in a computationally efficient manner.

The initial MDOPT system release provides a 3-D vehicle shape (outer mold line, OML) design optimization capability for wing alone, wing in the presence of a body, or wing with body and a single longitudinal controller (canard or conventional horizontal tail). Global or local direct driven design optimizations may be completed using a variety of multidisciplinary objective and constraint functions, including mission performance metrics, aerodynamic characteristics, weight trends and stability and control characteristics.

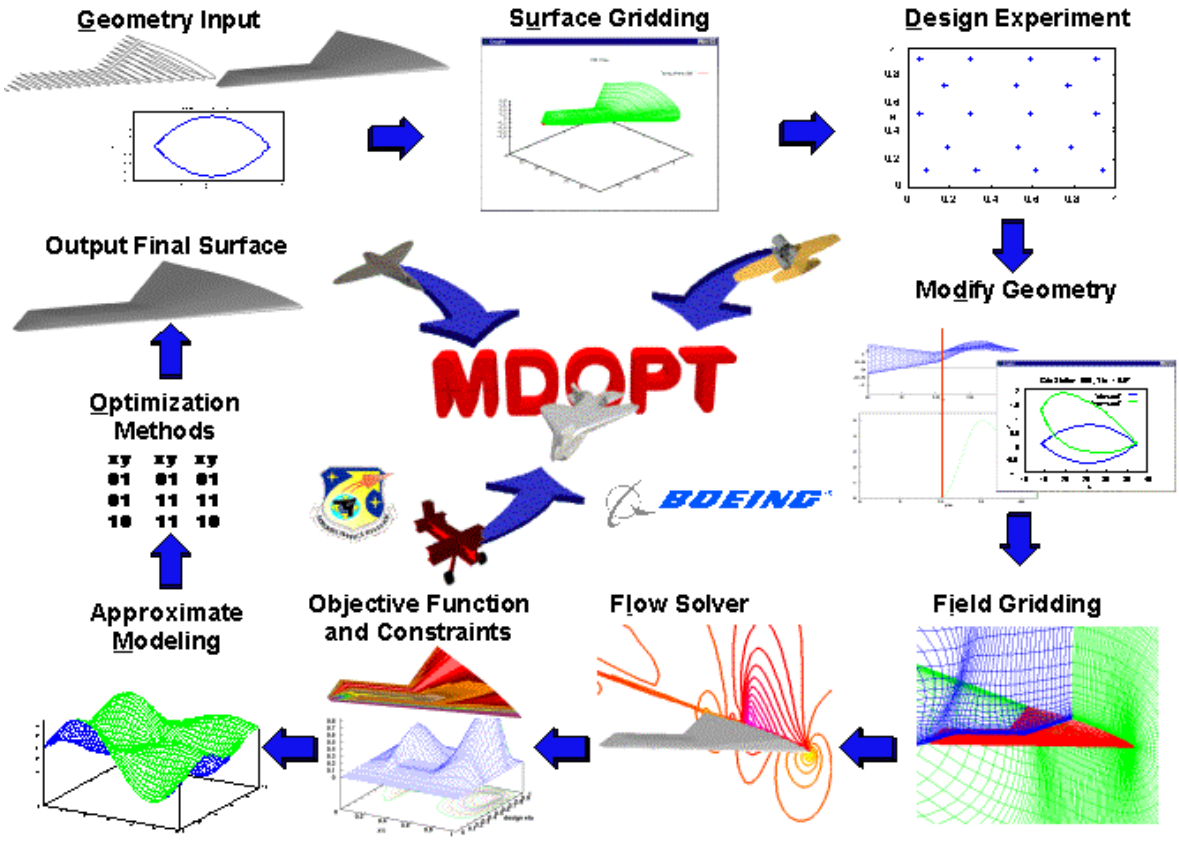

Figure 2. MDOPT Process Overview

II. Inter-domain Communication Facility (ICF)

The ICF shown in Figure 5 is an enterprise-level framework that supports loosely and tightly coupled distributed network access to engineering information and analysis services. The ICF supports arbitrary user-defined workflows and complex binary data structures. It interoperates well with interactive processes initiated on SMP and clustered computing environments, as well as, processes initiated by batch queuing systems such as Portable Batch System (PBS[7]).

The architecture and associated technology selections for the ICF were mainly driven by the following requirements: 1) Open-standards based (2 Interoperable with other standards-based implementations 3) Handles arbitrary text or binary data efficiently 4) Supports flexible workflows including hierarchical and asynchronous processes 5) Medium risk 6) Robust operation during long running processes 7) Multi-user support 8) Allowance for man-in-the-loop interactions 9) Low cost 10) Ports to all supported Unix platforms and 11) Dovetails easily with the rest of the MDOPT system.

A. Technologies Selection

1. Middleware Selection

Given that in a large distributed engineering environment each discipline maintains specialized domain knowledge and might prefer control over their processes, one of the design goals of the ICF was to allow access to these services without forcing each organization to relinquish control over their processes. The ICF is built upon the

foundation of the flexible, widely adopted, and stable CORBA[4] standard developed over the last decade. Presumably by using a standard like CORBA engineering and finance departments providing information and analysis services would be more likely to make the investment in exposing their services without the fear of vendor-lock-in or continuous retooling caused by being early adopters of various fledgling communication frameworks. During the development of the CORBA standard in the 90's, CORBA implementations had a history of being expensive, non-interoperable, and overly complicated. With the stabilization of the standard in the late 90's and the current availability of many compliant free and commercial implementations[8] (see Figure 4) with bindings for C, C++, Java, Tcl, Python, Perl, and powerful developer-friendly scriptable solutions these growing pains have practically disappeared. A current example of interoperability between three different ORBs written in three different languages is presented in Appendix A.

CORBA uses a fairly efficient binary protocol called GIOP. GIOP is transport independent. The TCP/IP implementation is called IIOP. The ORB takes care of the tedious details involved in converting machine dependent data structures to flattened versions in the standardized binary format defined by the GIOP specification. It also takes care of the basic network communication between client and server. A benefit of using a binary format is that there is no impedance mismatch for text and binary data; it is all treated the same way. Any data structure that can be described by IDL can be easily transmitted. Blobs of text or binary data can be described as binary sequences. Small blobs can be passed with one method call while large blobs can be passed using a file iterator such as shown in Figure 3.

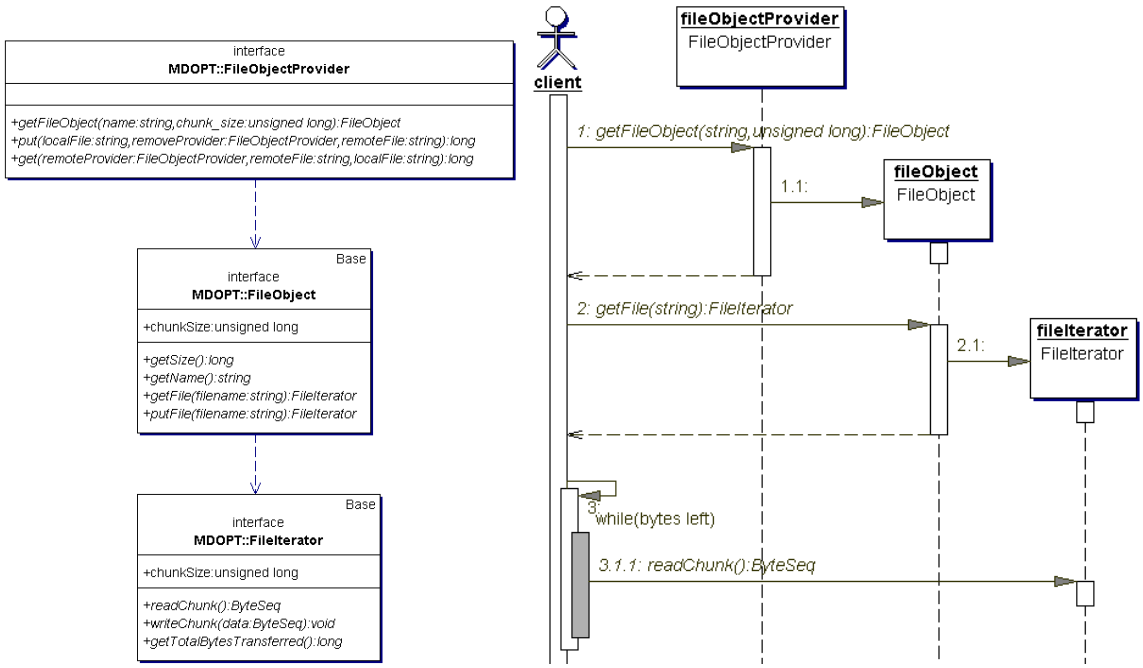

Figure 3. A class and sequence diagram for an example file iterator design

The CORBA standard also defines a number of vertical services such as COSS Naming, Interface Repository, Event Service, Notification Service, Trading Service, etc. Many of the available CORBA ORB implementations also bundle a number of the standard services with the ORB. One of the byproducts of standards-driven development is that the best of breed implementation of a given service implementation can be chosen without much or any retooling of the infrastructure. The ability to switch out implementations can be important when a given project begins to scale beyond the performance capabilities of a given implementation.

The CORBA ORB implementations chosen for the current ICF are MICO[5] and Combat[3]. MICO has proven to be very portable and reliable. We have successfully ported MICO and Combat to the Cray and have had success running on Solaris, IRIX, Linux and Windows operating systems. Interestingly, MICO has reported to run on a PocketPC and the embedded Linux OS uCLinux[9]. MICO is also well supported both commercially and by the user community. Combat was selected since it was the best way to interface with Tcl using CORBA. As shown in Appendix A, it is also very easy to develop a CORBA client/server applications using Combat.

ORB Name	Web Location	Programming Language Bindings	CORBA Specification Compliance	Open Source	Free
MICO	http://www.mico.org	C++, Tcl	2.3	Yes	Yes
Combat	http://www.fpx.de/Combat	Tcl	< 2.3	Yes	Yes
Orbit	http://orbit.sourceforge.net	C, C++, Tcl, Perl, Python	< 2.3	Yes	Yes
Fnorb	http://fnorb.sourceforge.net	Python	> 2.0	Yes	Yes
Tao	http://www.theaceorb.com/	C++	2.6	Yes	Yes
OpenORB	http://openorb.sourceforge.net	Java	2.5	Yes	Yes
JacORB	http://www.jacorb.org	Java	>2.3	Yes	Yes
JDK 1.5	http://java.sun.com	Java	< 2.3	Yes	Yes
OmniOrb	http://omniorb.sourceforge.net	C++, Python	2.6	Yes	Yes
Orbacus	http://www.orbacus.com	C++, Java	> 2.5	Yes	No
Visibroker	http://www.borland.com	C++, Java	> 2.5	No	No
Orbix	http://www.iona.com	C++, Java	> 2.5	No	No

Figure 4. Examples of current commercial and open source CORBA implementations

2. Primary Language Selection

The MDOPT project is an incremental extension of the 3DOPT[10,11] project that began in 1995. The GUI for 3DOPT was developed using the Tcl language; the object oriented extension incrTcl[12] and the Tk and Tix widget sets. Tcl/Tk was selected for the GUI development given the rapid development times demonstrated for Tcl/Tk-based GUIs; the accompanying rich set of extensions and widget sets; the immediate portability of the GUI to the various Unix platforms and the ability to catch and display errors without crashing. The resulting GUI was also easy to modify by aerospace engineers who might not have had training in X-Windows and C++ programming. Tcl still enjoys significant support in the user community and it can be extended using C, C++ or Java[13]. Given that the GUI had to communicate with the ICF, the successful experience of our Tcl development over the years, and the availability of Combat, it was natural to pick Tcl as the language for the rapid development of many of the ICF services. Given that there are CORBA bindings available for C++ and Java as well, performance-driven or distributed web-based applications using applets or Java Web Start can be developed in the future as the need arises without reworking the framework.

B. Architectural Overview

Referring to Figure 5, the ICF is a collection of specialized services that communicate with each other using CORBA. It leverages three of the standard vertical CORBA services bundled with the MICO CORBA distribution: Naming Service, Event Service and Interface Repository.

The COSS Naming Service is used as a “white pages” style lookup of the currently running processes. The processes are organized hierarchically and can be keyed off a given process owner ID. The GUI only displays those services associated with the current process owner.

The interface repository (IR) stores the functional interfaces and data types defined by the IDL of the available ICF services.

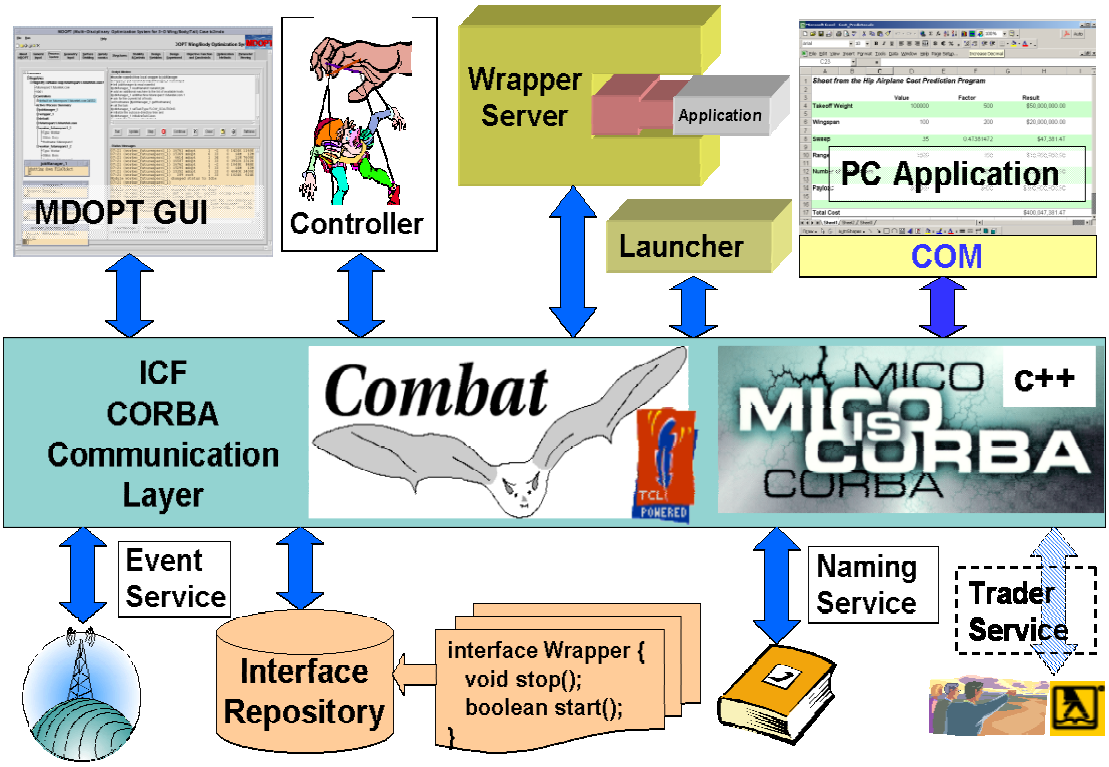

Figure 5. Architectural overview of the ICF

The GUI dynamically queries the IR for these definitions and provides them graphically to the user to allow him to define a workflow in the script editor. The ICF clients and servers built using Combat retrieve the required interface and data type definitions at run-time from the interface repository. The IR is also relied upon for code generation of incrTcl skeleton code for new ICF services.

The CORBA Event Service[14] provides an asynchronous publish and subscribe messaging capability similar in behavior to a radio station broadcast. Clients (providers) can push information to a given channel on the event server and numerous interested consumers will receive the information. A separate event channel is created for each process owner and design case combination. The ICF leverages this capability to allow longer running processes to broadcast their status and analysis output information to multiple GUI clients. A given GUI can tap into the information stream at any point in time to view the current state of the MDO processes. The system was designed such that no state information was stored within the GUI thus allowing the GUI to be shutdown and restarted at will without any impact on process execution; the GUI merely provides a portal to the executing process flow. Multiple GUIs can also be connected simultaneously from different computers. Currently, it is the GUI's responsibility to filter the messages sent by the various ICF services associated with the design case prior to displaying them to the user.

Most of the ICF processes are designed to be autonomous CORBA compliant servers. This feature makes it easy to choreograph arbitrary process flows between them by writing client code using any of the available CORBA language bindings. As stated earlier, an innovative open-source binding for the Tcl language called Combat, written by a German developer named Frank Pilhofer, was utilized to allow the user to script process flows from within the MDOPT GUI as shown in Figure 6. By leveraging a mature full-featured language like Tcl, complex process flows are flexibly supported in a more natural way than perhaps a custom language solution might allow.

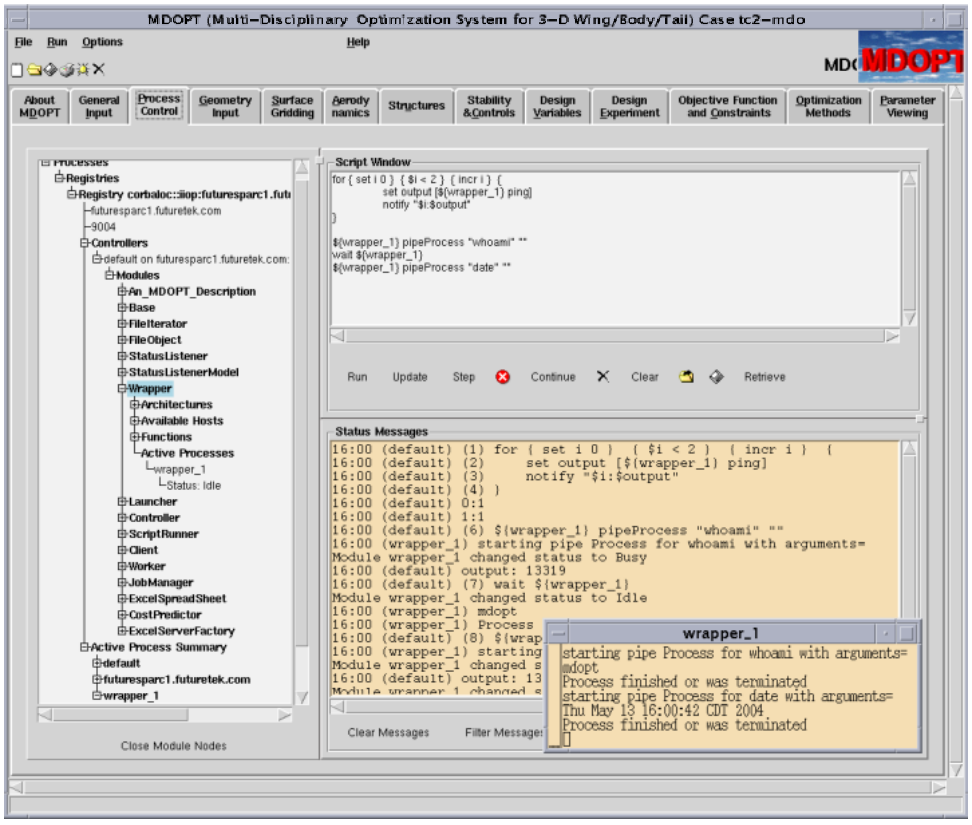

Figure 6. An example of defining a simple process flow in the ICF GUI

Combat was also used to create various ICF server processes. For instance, a service called the "Wrapper" was developed to allow loosely coupled integration of existing engineering applications. As depicted in Figure 7, this service executes and communicates with the desired application via a UNIX pipe. The Wrapper allows remote control over the life-cycle of the launched application, redirects the standard output from the launched application to the event server for broadcasting to the GUI, provides file transfer capabilities to any other Wrapper and status information to be queried and broadcast.

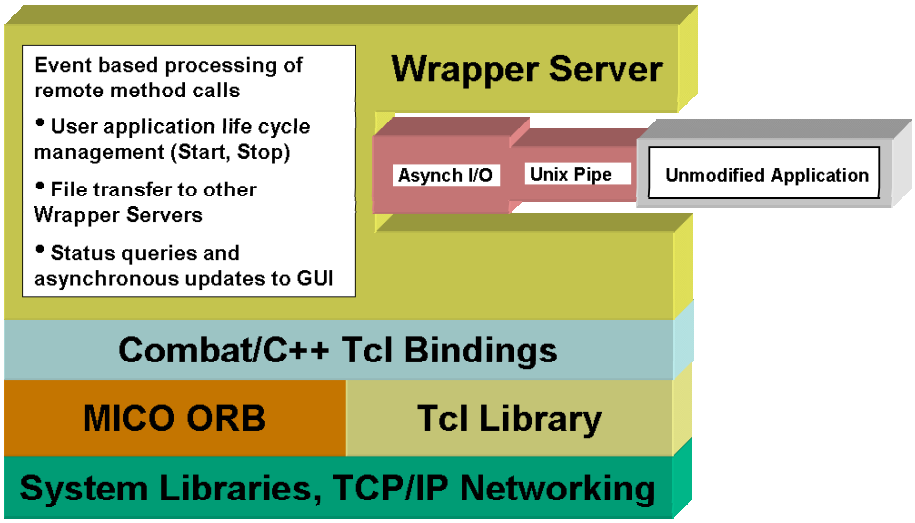

Figure 7. Wrapper Server

Other ICF services shown in the class diagram in Figure 8 developed with Combat are the Controller, Launcher, and Job Management system. The Controller provides a convenient façade interface that exposes various factory methods for generating instances of ICF Script Runners, Launchers, Wrappers and other services. All services can be started manually from the command line as well.

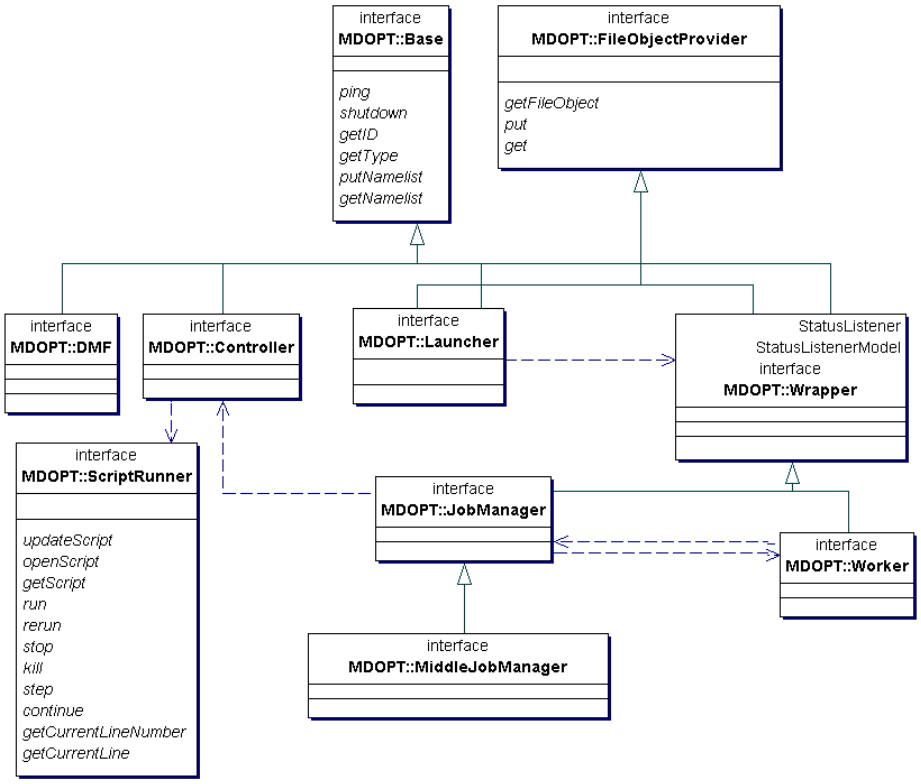

Figure 8. Class diagram of ICF services

The ScriptRunners are lightweight processes used by the GUI to execute and step through the ICF scripts generated by the user. The GUI interacts with the Script Runner via the ICF. The Script Runners execute within their own thread within the Controller Server to avoid blocking the entire server during a long running ICF script. One of the benefits of running the scripts in a server process rather than within the GUI is that the GUI can be disconnected as soon as the script is initiated. An ICF script can also be embedded in a non-GUI sub process and executed in-line or via a ScriptRunner. (A serendipitous use of the Script Runner is a simple multi-user chat capability between MDOPT GUIs.)

The Launcher Service is a simple server that starts registered ICF services only on the machine on which it is running. The Launcher Service is started on the remote machine by the Controller via standard remote shell or manually by the user on the command line. A benefit of this service is that remote shell does not need to be available on all the machines accessible to the ICF such as Windows computers or those with remote shell disabled. Starting up new processes is also faster than using remote shell. The Launcher only starts trusted ICF processes that are present in a configuration file.

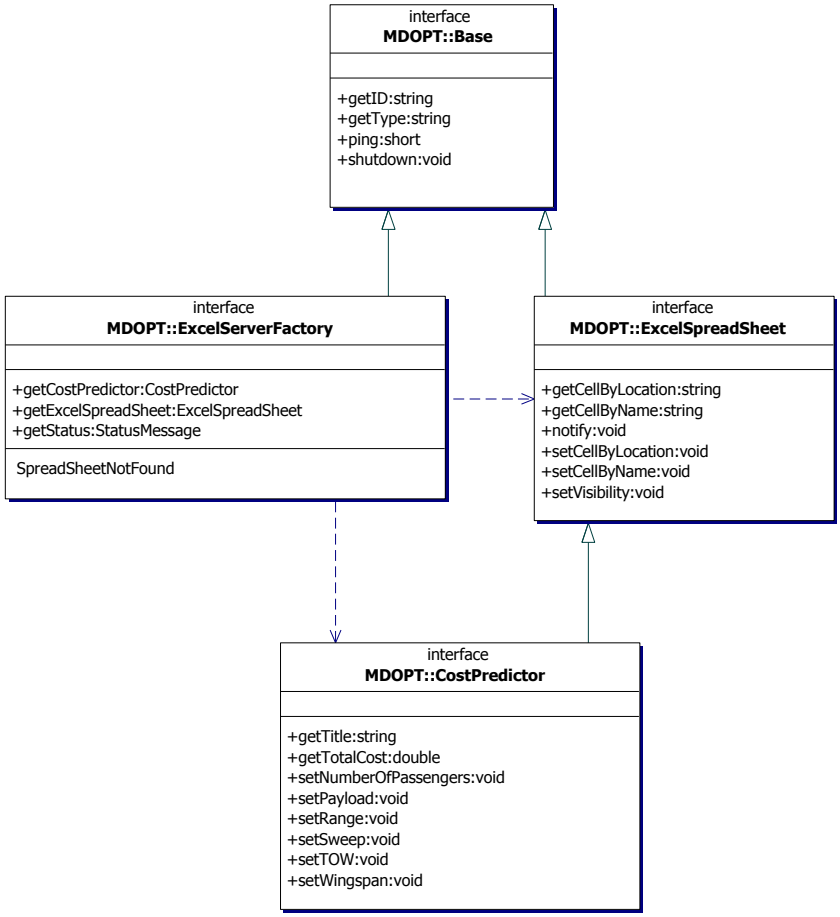

Figure 9. Class diagram of Excel Services

Microsoft Excel spreadsheets are significant sources of information from disciplines such as finance or airplane weights. These spreadsheets can be accessed through the ICF in a loosely coupled or tightly coupled fashion. (A Tcl extension called tcom[15] is used to communicate programmatically with Microsoft Windows programs such as Excel via COM.) In the first case, an instance of a generic ExcelSpreadSheet service that instantiates the specified spreadsheet is created through the ExcelServerFactory service that is running on the PC where the spreadsheets are located. The spreadsheet's cell data are accessed by row and column addresses programmatically from within an ICF script. This approach has the advantage that spreadsheets can be quickly integrated, but it can lead to a brittle integration since the ICF scripts have to change any time a given field is moved to another cell location or it's name changes.. Another supported approach that provides a tighter integration and more intuitive interface with the spreadsheet is to create an extension of the ExcelSpreadSheet service through inheritance that exposes the information via specific "getter" and "setter" methods with names patterned off the desired fields or cell aliases. This allows the new service to shelter the developers of ICF scripts from structural changes in the spreadsheet. Furthermore, the new service and the more intuitive methods will appear in the GUI once the new IDL is loaded into the IR. The class diagram for this example is shown in Figure 9**.** An example of interacting with spreadsheet using this latter approach is shown in Figure 10 and the accompanying sequence diagram is shown in Figure 11**.**

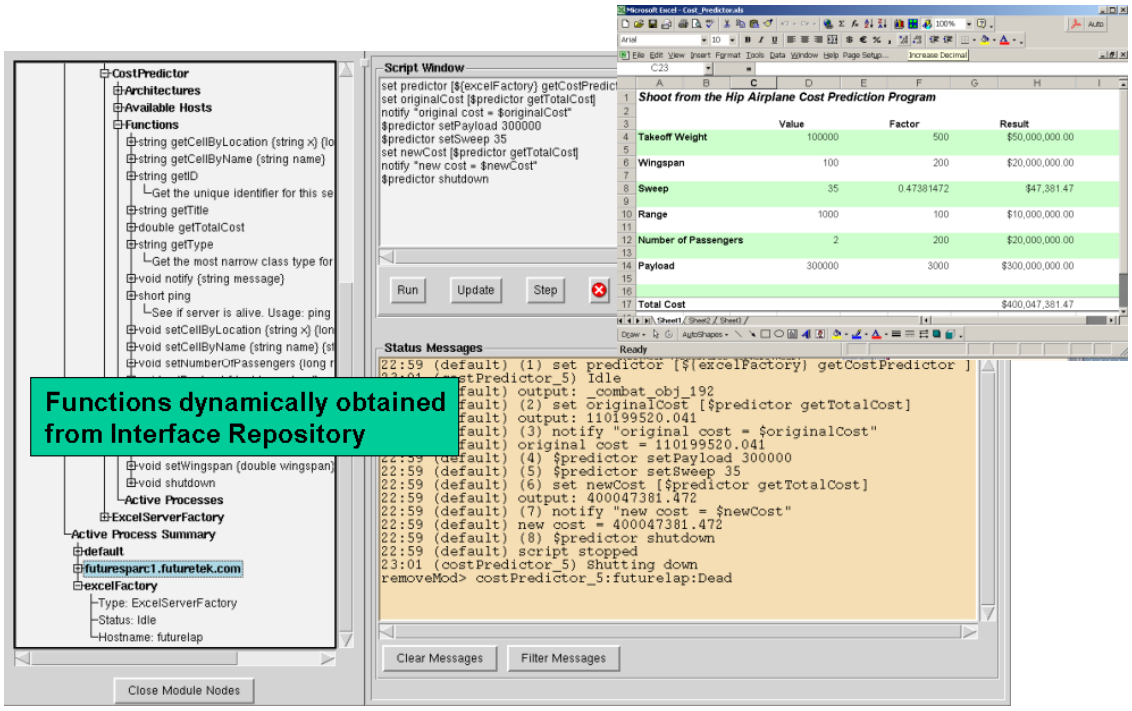

Figure 10. An example of interacting with an MS-Excel spreadsheet

Figure 11. A typical interaction with the Excel services

C. Life cycle management

It would be a hardship for the user to have to manually shutdown the many associated services running on various computers if he decided to cancel an entire process flow for example. It would be easy to miss a few services resulting in many orphaned processes eventually. The ICF has been designed to manage the life cycle of the services in a practical way. As a general rule, any process that creates other processes shuts down those processes before shutting itself down. For instance, a Controller is associated with a given design case. If the Controller is shutdown it first shuts down all processes associated with the design case such as Launchers and Wrappers before shutting itself down. Any ICF service can also be shutdown from the GUI or by issuing a "kill –1" on the process ID or by pressing <Cntrl-C> on the command line for services being run interactively. When a process is notified to shutdown, it notifies all subscribers to the Event Service that it is shutting down and removes itself from the Naming Service registry to eliminate stale handles.

If a service terminates unexpectedly (such as if somebody issues a "kill –9" on the process ID) it will not get a chance to unregister from the Naming Service and a stale entry will be left in the Naming Service. To handle this situation, a given ICF client, such as the ScriptRunner or GUI, will initially test each handle retrieved from the Naming Service by pinging it for liveliness prior to using it. If the ping fails, the handle is unregistered.

D. Starting a new ICF service

The sequence diagram depicted in Figure 12 represents a typical workflow for ICF service creation. The steps are described as follows:

1. The Client (typically the GUI) asks the Naming Service for a reference to the Event Service Channel Factory by name. For the ICF, the initial Event Service reference is always called "EventChannelFactory" in the Naming Service.
2. Once the Client has a reference to the Event Service, it tells the Event Service to register the Client as an interested party for all future system messages. (This actually implies that the Client will act as a server to the Event Service.)
3. The Client asks the Naming Service for a reference to the Controller service.
4. Once the Client has the reference, it tells the Controller to start a new "Wrapper" process on Server A.
   - 4.1. If a Launcher does not exist on Server A, the Controller creates one called Launcher_A.
   - 4.2. Launcher_A registers itself with the Naming Service so it can be located easily in the future.
   - 4.3. Once Launcher_A is started, the Controller tells it to start a Wrapper on Server A.
   - 4.4. Launcher_A starts a new Wrapper called Wrapper_1
5. Wrapper_1 also registers itself with the Naming Service so other processes can locate it easily in the future.
6. Wrapper_1, after looking up the Event Service, registers itself with the Event Service as a producer of messages and sends a status message to the Event Service
7. Having received a reference to the new Wrapper via the Controller in step 4, the Client tells Wrapper_1 to start a process. Notice that once a reference is obtained to a given service that methods can be invoked without knowing where the service is located.
8. Interactions with the Interface Repository are not explicitly depicted for the sake of clarity. Combat uses the Interface Repository to look up every interface of the service that it is calling prior to calling methods on the service. In addition, every Combat service looks up its own interface from the Interface Repository during initialization.

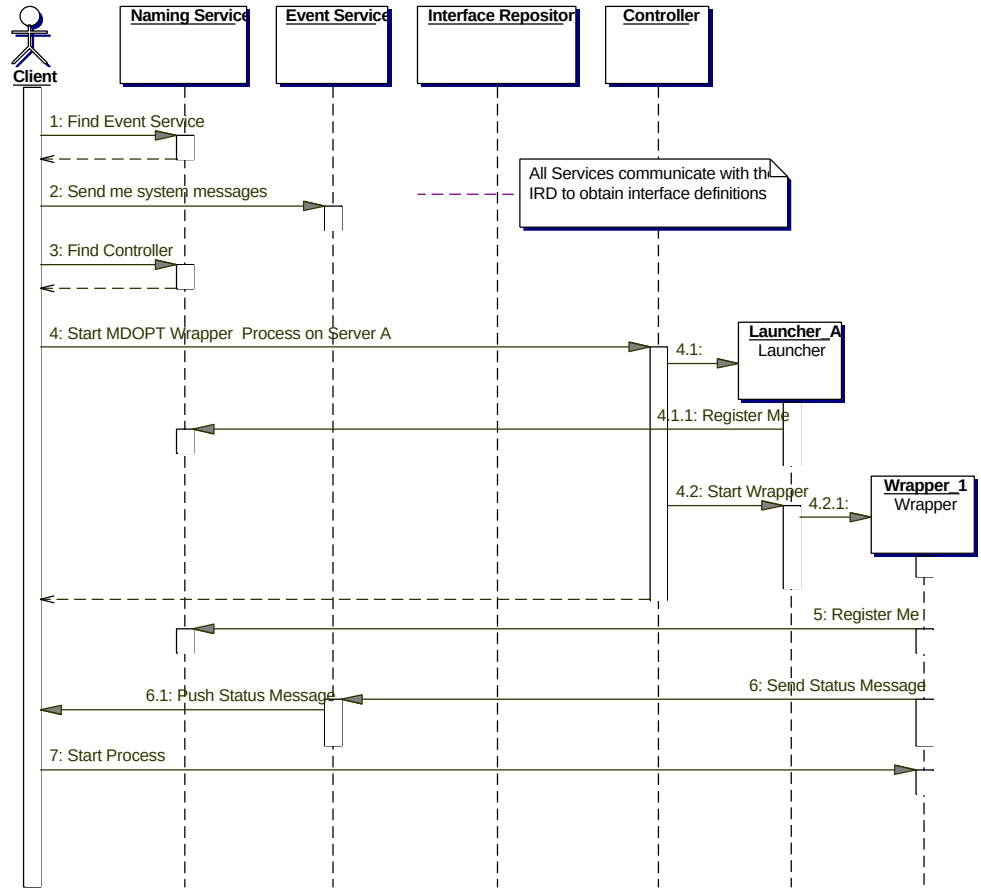

Figure 12. Service creation workflow

E. Job Manager System

The MDO process requires a large number of computer aided engineering solutions, most of which can be run in parallel. In the past, the user was responsible for doling out the solutions to the various computers available to him. It was difficult to know how to best distribute the solutions for the most efficient turnaround and to manage the associated bookkeeping. A distributed job management system was developed to alleviate this problem using the facilities of the ICF. This capability allows a client, such as a GUI, to start a Job Manager process that efficiently parcels out jobs from a queue that it manages to workers started by the Job Manager or started manually on selected machines. Once initiated, the workers request a task from the Job Manager. The workers continue to request jobs until the queue is empty after which they are terminated. This approach allows efficient use of heterogeneous computing resources including computers with managed batch queuing systems such as PBS and Linux clusters.

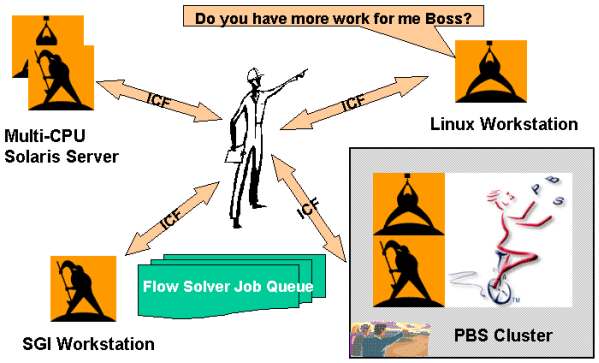

Figure 13. JobManager

Linux clusters added a communication complication for the JobManager and other interested ICF clients. The typical Linux cluster has a gateway node that can communicate with the internal nodes in the cluster and with computers on the intranet. The internal nodes can contact computers on the intranet, but they cannot be contacted directly from computers outside the cluster. Currently, MICO and Combat only support unidirectional IIOP

communication and not bi-directional IIOP. This current state of affairs led to the solution shown in Figure 14. A MiddleJobManager runs on the gateway node and acts as a proxy for the Workers running within the cluster. Any communication with the Workers must go through the MiddleJobManager. This communication is transparent to the GUI and other clients wishing to contact the Workers directly since the MidddleJobManager registers proxy Workers in the NamingService for each Worker that it manages. Communications with the proxies simply get delegated to the actual Workers. Also, when the Workers ask their MiddleJobManager proxy for the next job, it simply passes the request up to the JobManager that is managing the central queue for all the Workers.

The jobs added to the JobManager's queue are themselves Tcl scripts that are executed by the Worker using Tcl's "`eval`" command without any prior knowledge of what the script does including interacting with other ICF services to retrieve input files for instance. When adding the jobs to the queue, configuration constraint attributes can be also specified such as allowable computer configurations. Each job can also be specified as a sequence of dependent jobs.

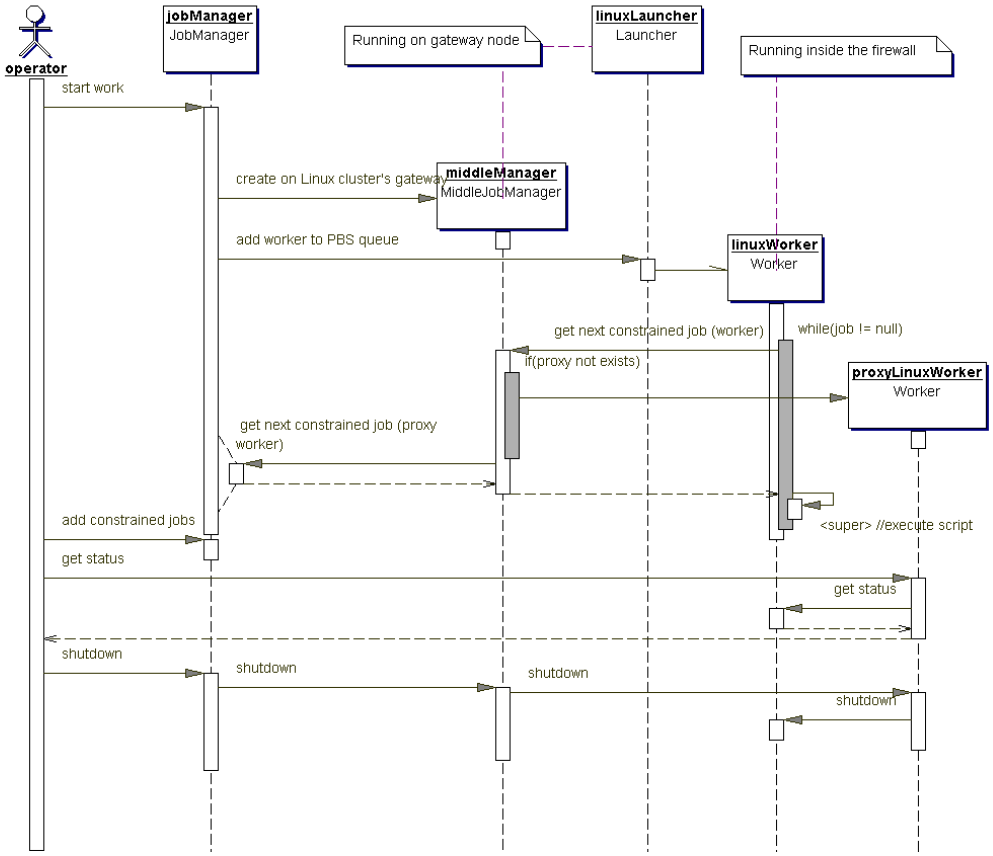

Figure 14. Communicating with workers within the firewalled Linux cluster

F. Deployment of ICF Services

To clarify how a network of ICF services would appear on the network, a sample deployment is shown in Figure 15. For this configuration, the core CORBA services are running on a Linux workstation A. Other services are deployed across the network on SGI, Cray, Sun and Windows machines. All of the ICF services depend on the core services running on Workstation A. These core services can be shared by all MDOPT users or each user can start his own set of services. Notice that on the Windows machine, that a Launcher is not running. This is meant to illustrate a case where a user started a server (the ExcelFactory server) manually or with a system startup script. Due to the networking issues discussed earlier on the Linux cluster, the Launcher and the MiddleJobManager must run on the gateway node. The workers run on each individual node in the cluster.

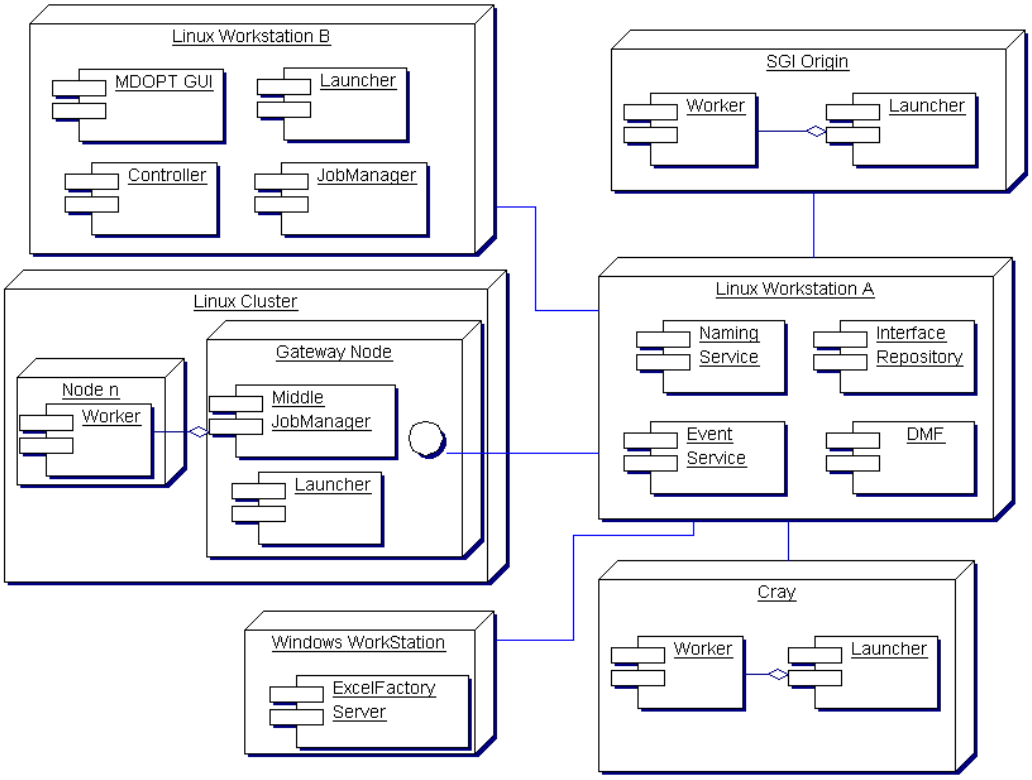

Figure 15. Sample Deployment of ICF Services

G. Overview of new ICF module creation

To create a new ICF service, the IDL describing the service must be generated. To work seamlessly within the MDOPT environment, the new interface should extend MDOPT::Base. If the implementation of the new service is written in Tcl, the skeleton and some boilerplate code for the new service can be auto generated based on the new IDL loaded in the IR. The skeleton will inherit ICF base functionality from the MDOPT::Base class implementation. The new service implementation must allow the specification of the event channel ID, the process name and process owner as arguments to the program.

A simple input form for the MDOPT GUI for the new module's inputs can be automatically generated at run time based on an XML definition of the metadata for the input variables as shown in the example in Figure 16. The description field is shown as "CDATA" to allow descriptions to be written in HTML in the future. The user can also create a custom form if desired if the auto-generated form is not sufficient. The associated data to populate the form is extracted from the master control document based on the module name. There are some richer implementations[16],[17]of this concept available now in the industry that are good examples of how far this concept can be taken.

Once the IDL is loaded into the IR, the new module's methods will appear in the GUI after a new startup. The new service must also be registered with the Launcher service as a trusted application by modifying an associated file. This last step isn't necessary if the service does not need to be started by the Launcher.

```
- <variable_definition>
  - <module name="MyModule">
    - <group name="group1" label="My Group Label" suggestedHeight="100">
      - <file name="myFileVariable" label="My File Variable" default="myfile.dat">
          <description>Description of the file input</description>
        </file>
      - <enumeration name="optionVariable" label="My Option Variable" default="option1">
        - <options>
            <option value="option1" label="Option 1"/>
            <option value="option2" label="Option 2"/>
            <option value="option3" label="Option 3"/>
          </options>
          <description> description of the optionVariable input</description>
        </enumeration>
      - <boolean name="myBooleanVariable" label="My Boolean Variable" default="true">
          <description>A description of my boolean variable </description>
        </boolean>
      - <double name="myDoubleVariable" label="My Double Variable" default="20">
          <description>A description of my double variable </description>
        </double>
      - <integer name="myIntegerVariable" label="My Integer Variable" default="1">
          <description> integer description 2 </description>
        </integer>
      - <string name="myStringVariable" label="My String Variable" default="some text">
          <description>A description of my string variable</description>
        </string>
      </group>
    </module>
  </variable_definition>
```

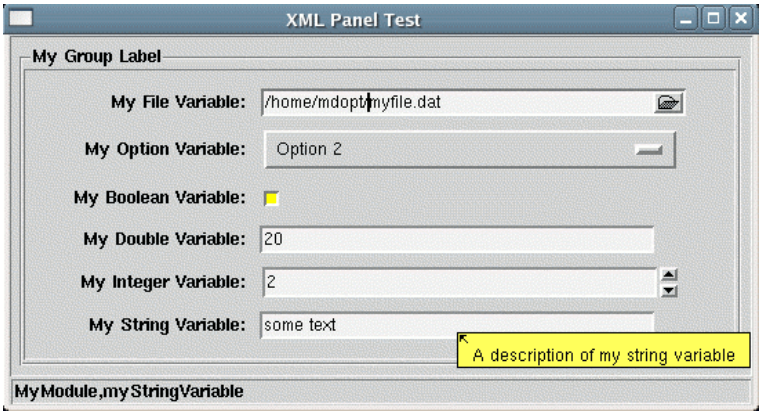

Figure 16. An example of an XML description of the meta data for module inputs and the associated auto-generated GUI

H. The Benefits of Dynamic Parsing and Invocation

1. Combat's CORBA use of DII and DSI

In the past, CORBA's fixed binary data structures described through IDL have had the problem of being brittle and not easily versioned. This problem was because IDL doesn't support versioning very well and typically the stubs and skeletons for the clients and server code were statically generated, compiled and linked. Any time even one field was added to an existing IDL struct or valuetype, all of the clients and servers would have to be upgraded in the field to avoid marshalling and parsing errors. The CORBA standard does support reflective-style dynamic invocation and marshaling via the Dynamic Invocation Interface (DII) and the Dynamic Skeleton Interfaces (DSI) that leverage the IDL definitions stored in the IR server. Using this mechanism, the binary data stream can be marshaled and parsed based on the current definition of the data stream obtained from the IFR. Using this mechanism, Combat can determine how to parse a given data stream at run time based metadata retrieved from the IFR.

2. Recent developments with CORBA reflection

Recently, Steve Vinoski of IONA proposed[18] and passed a standard for CORBA implementers to support this same reflective capability but without the requirement for an interface repository[19]. Each object reference can be queried at run time for its interface information in either XML or the in the format described for the interface repository by the CORBA standard. The new Reflection interface is very simple:

```
#pragma prefix "omg.org"
module Reflection {
     exception FormatNotSupported {};
     interface IFRProvider {
       any omg_get_ifr_metadata () raises (FormatNotSupported);
       string omg_get_xml_metadata () raises (FormatNotSupported);
     };
};
```

Frank Pilhofer has included this capability in the current Tcl version of Combat as well thus eliminating the requirement for the interface repository server. (The GUI would still use the IR to allow user to interactively create scripts in the script editor.) MICO's recent 2.3.12 release also supports the new IFRProvider interface. This reflective capability should give CORBA users the ability to write code that is much more flexible and adaptable. For instance, generic parsers can be created that will parse the binary stream based on the current definition of the data records retrieved from the interface. If the program expects an older record, any additional data provided in the record would be simply ignored without any unmarshalling errors.

On a side note, ASCII data formats like XML or Fortran-style namelists can be passed around just as easily as binary data using CORBA's binary IIOP protocol if desired. This is done for example with the namelist file in the ICF when it can be more flexible to send the entire set of inputs rather than stripping out the minimum inputs required and generating a set of associated IDL definitions. The namelist inputs are passed using a sequence of name/list of value pairs defined in IDL as:

```
typedef sequence<string> NameValues; # value data for each variable
typedef sequence<string> NameTypes; # type data for each variable
void putNamelist ( in  NameValues nameValues , in NameTypes nameTypes );
```

These lists can be easily converted to a Tcl hash array using: `array set values $nameValues`
XML, of course, could be passed in a similar way.

III. The Data Management Facility (DMF)

The DMF is a framework that allows networked access to a centralized relational multi-user database. The current DMF is built upon the free open source relational database MySQL and a Tcl extension called MySQLTcl[20]. MySQL is a fast, networked, multi-user database that supports most of the SQL 92 standard. MySQLTcl provides the Tcl bindings to the MySQL communication library.

The DMF architecture allows data from non-automated and automated engineering processes to be included in the optimization process from anywhere on the network. Currently, the DMF is used to store all of the scalar and vector data needed by the different MDOPT processes. For instance, objective and constraint values required by the response surface generation routines are calculated from the raw data stored in the database. Data can be stored and edited either interactively through the DMF GUI as shown in Figure 17 or via the Unix-style command line interface that can be embedded into shell scripts. The command line interface allows data stored locally in a file with a Fortran-style namelist format (i.e. x = comma delimited list) to be stored in the database and the data stored in the database to be retrieved locally to a namelist file. The DMF functionality can also be directly accessed programmatically from within Tcl scripts via the object-oriented DMF library or the ICF's DMF proxy.

The DMF represents the canonical form of the data used in the optimization process. This approach limits the creation of many to many translators required in a process where adaptors are written to translate the output of one program into the inputs for a second program. Translators for scalar and vector data stored in the database only need to know how translate to and from one format stored in the database from and to one particular program's format.

The DMF relies on a relational database. A typical relational database organizes data by columns in various interconnected tables. The entity relational diagram, ERD, shown in Figure 18 depicts the tables and their relationships to each other for the DMF database. All the desired computed data for the solutions are stored in one table called VarValues. This type of table structure was designed to allow the database to scale to any number of variables associated with data vectors of varying lengths without having to create new table columns for each variable. Since MDOPT allows the user to create variables on the fly, it was important that the database schema support new variables without requiring table structure changes. The design tradeoff is that SQL queries are a little more complicated and performance can be slower, but through the careful use of table indexes, data retrieval from the DMF is fast even with such a simple table structure.

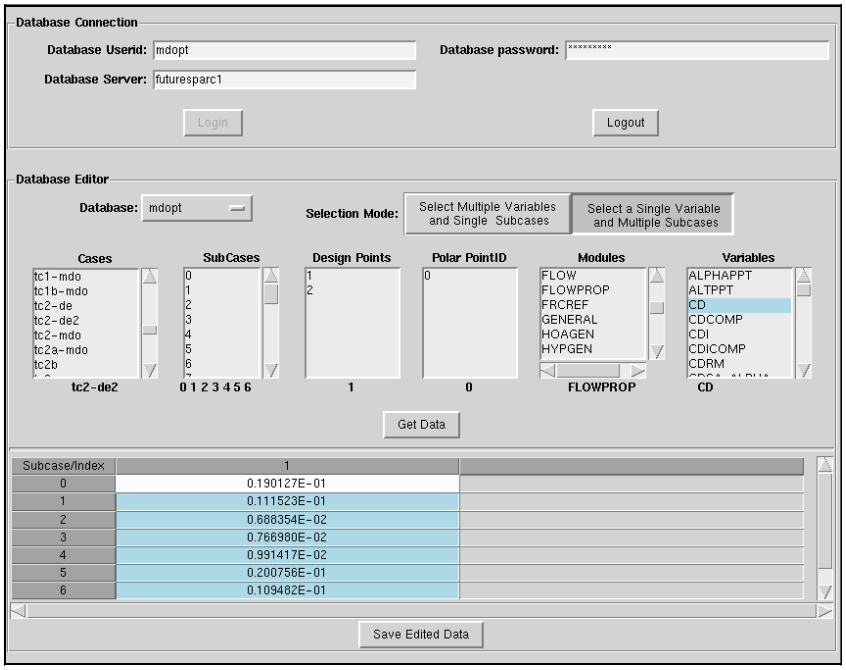

Figure 17. DMF GUI

Most of the other tables provide metadata associated with the computed data values. For example, the Variables table contains descriptions of each variable including its name, type, units, allowable range, and a human readable description. This information is available from the DMF GUI. The Cases table's columns provide a way to ID a particular solution. The current scheme reflects the domain-specific organizational structure of the data.

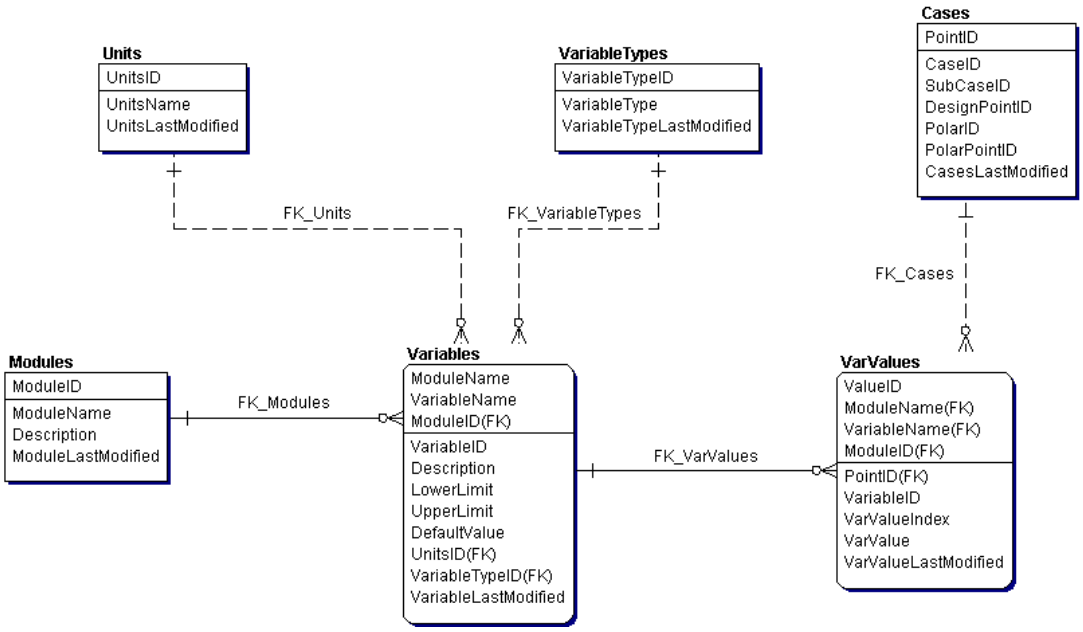

Figure 18. Database Schema for DMF

One recent addition to MDOPT is the ability to track the status of steps of the process in the database. Traditionally, a status summary was stored for each PointID in the file system. But, this had the disadvantage of not being easily accessible from non-NFS mounted computers, being slow to access for the 100's of cases being run and only recorded the status of the last step. The information is organized as shown in Figure 19. A given process is a number of steps. Each step has a unique name and a number. The number does not have to be unique in order to handle parallel steps. Step names can be process names in order to model a hierarchy of processes. A notes log table has also been included so that the users keep a journal of the design work.

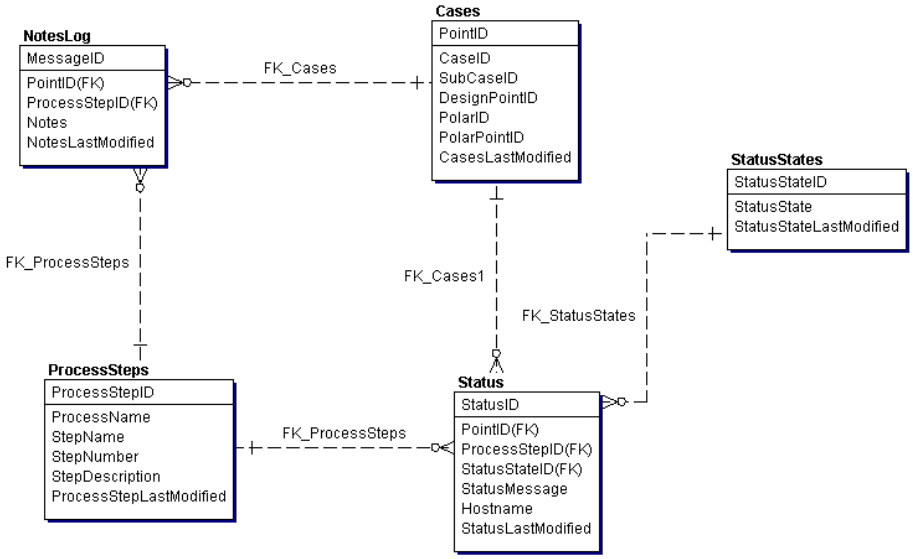

Figure 19. Database Schema for Status and Logging

A very useful feature of this capability is that the JobManager can query the status of a given job if it loses connection with a given worker before the worker had a chance to report to the JobManager that it was finished. It also allows for finer grain knowledge of where the process failed so that the process can be restarted from the last successful step.

Database access is accomplished in different ways depending on the application. It is accessed directly via a IncrTcl façade that utilizes the MySQLTcl library. A DMF CORBA server is also available that delegates to the database façade object to allow generic CORBA clients to access data without having to have intimate knowledge of the database or to load a given MySQL communication library.

IV. Bringing it all together - An Optimization Workflow

MDOPT supports two different automatic optimizer-driven processing modes: 1) direct gradient based and 2) surrogate model refinement based. With the first approach, the gradient-based optimizer driver will request object and constraint values and their gradients for a given design vector. With the second approach, the driver will request object and constraint values for a series of design vectors. We have found over the years, that a fully automatic and robust optimization process is difficult to achieve in practice especially for the type of problem that MDOPT is addressing. The complex engineering codes used to analyze the airplane configurations are long running, resource intensive and can have many failure modes. Many times if a solution fails for a given design vector, the inputs can be changed manually to remedy the problem. Or, if there is related intermittent resource problem such as running out of disk space, resources can be cleared manually. If a failure occurs, it would be preferable if the process would suspend execution and allow a human to try to fix the problem and resume the execution rather than failing completely and possibly wasting days of computing time.

The ICF features discussed previously provided the foundation for the of an optimization process flow allowing a man-in-the-loop interaction. The current incarnation of this process is illustrated in Figure 20.

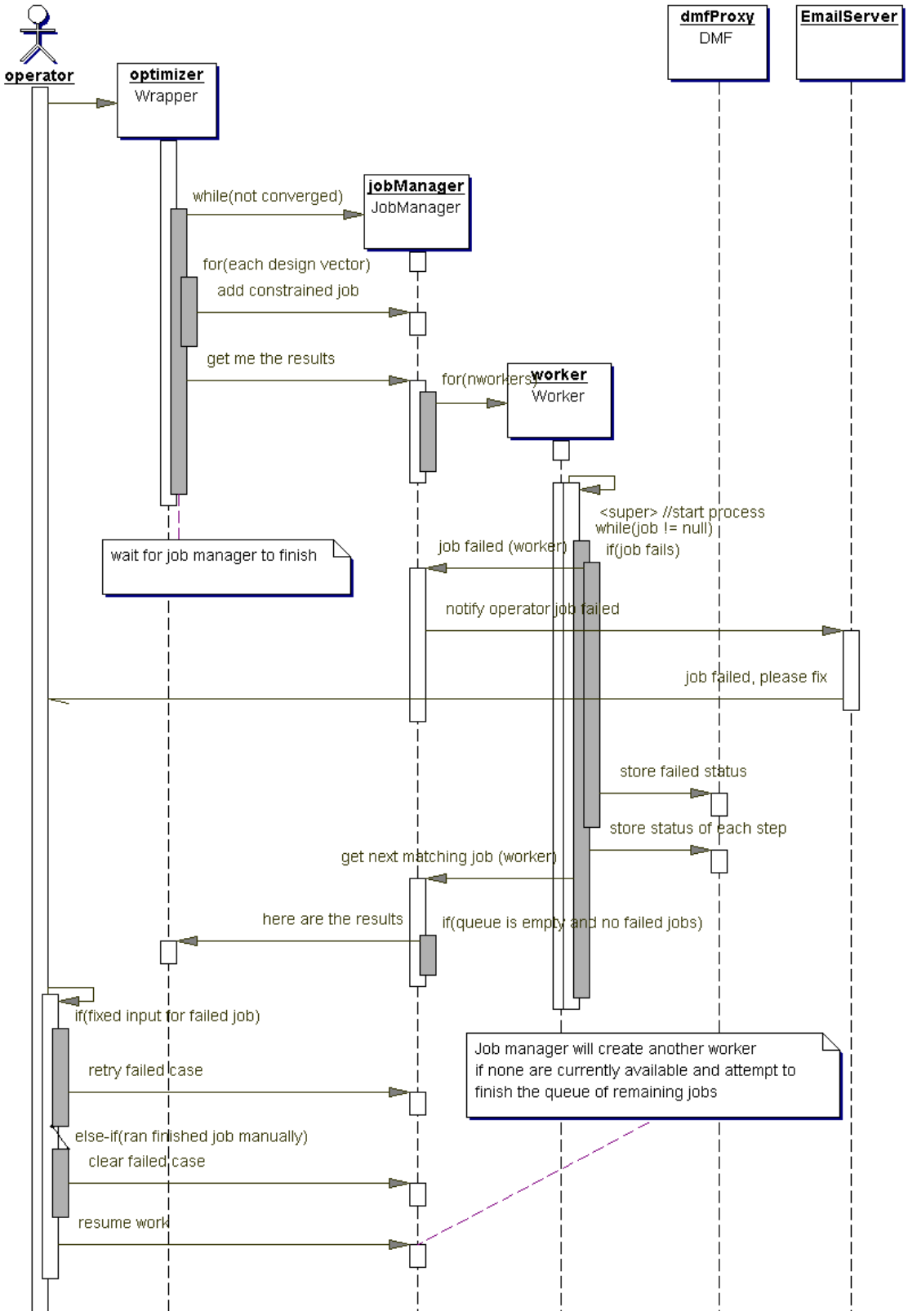

Figure 20. Man in the loop optimization process

As shown, when the optimizer driver requires function values for a set of n design vectors, it creates a new JobManager, adds n arbitrary jobs to the JobManager's queue, and requests that it produce the results. The JobManager then creates a set of workers on various machines specified by the user and manages the workers' requests for jobs. (The details of the actual process to create a worker have been eliminated for clarity.) If the worker detects that the job it was running failed to produce a valid solution, it notifies the JobManager that adds the job to a failure list and notifies the operator that there has been a job failure. The JobManager continues to process requests for jobs by the workers. Once the queue is exhausted, the JobManager terminates the workers as they ask for new work. If there are failed jobs in its list, the JobManager transitions to an idle state, otherwise it terminates itself and the optimizer driver reads the desired results from the database and continues to solve the problem. If the operator has been notified of a failure, he can take a number of actions to rectify the problem: he can run the solver for the particular design vector manually and then clear the job from the JobManager's failed job list; he can modify the inputs for the failed job and then tell the JobManager to retry the job; or he can choose to abort the entire process by shutting down the parent process. The ICF GUI gives the operator full freedom to query and modify the queue state of the JobManager as required, start additional workers or terminate existing ones, and retrieve file and state information from the workers as a given job progresses.

V. Future Work

The current ICF does not incorporate a robust security model yet. This has not been a requirement for the framework since all of the work is done within the confines of the corporate firewall. To communicate beyond the firewall, the combined used of standard portable interceptors to authorize access to a given method and secure sockets layer to encrypt the data would be used. We are also investigating the use of bi-directional GIOP or a full-fledged IIOP proxy to provide a more complete solution to simplify Linux cluster firewall communication. A more reflective Excel Server could be written that would allow the user to dynamically explore and access fields in an arbitrary spreadsheet through the ICF GUI. A graphical workflow editor based on one of the open source workflow projects will continue to be investigated. Currently all status and logging events are sent over one CORBA event channel for each major case. Although the user can filter the messages he wants to see at the GUI, it would be more efficient to filter the messages at the server using separate event channels per sub case or perhaps moving to the CORBA Notification Service instead that supports server-side filtering. The new CORBA reflection standard is compelling from a IDL versioning standpoint. As part of our research, we are going to investigate the advantages of using this new approach for unmarshalling of the data streams.

VI. Conclusions

The MDOPT system represents a major step forward in the development of design optimization capability. The standards-based ICF and DMF services give MDOPT cost-effective upgrade paths and flexible integration possibilities. The flexibility of the system has been demonstrated through actual implementations of complex workflows. The simplicity of development of the Combat scripting model for CORBA client/server applications has also been shown.

Appendix A – Examples of three CORBA ORBs using three different languages interoperating

Figures Figure 21 and Figure 22 are screenshots of two different interface repository browsers written in two different languages, Java and Tcl respectively, communicating with a CORBA server written in C++. The Java implementation was written using the open source JacORB ORB, the Tcl implementation used Combat, and the CORBA server was implemented with MICO. This example is meant to demonstrate that many of the interoperability issues between CORBA ORBs have been solved with modern implementations. There are still some interoperability issues for some of the more advanced security and communication (e.g. bi-directional IIOP) models, but they are being continually incrementally improved.

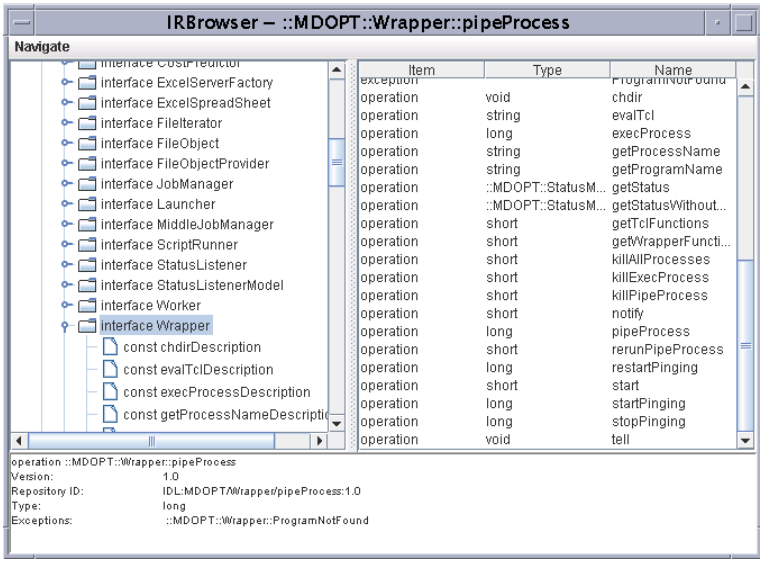

Figure 21. A Java IR browser client using JacORB communicating with the C++ ORB MICO's IR

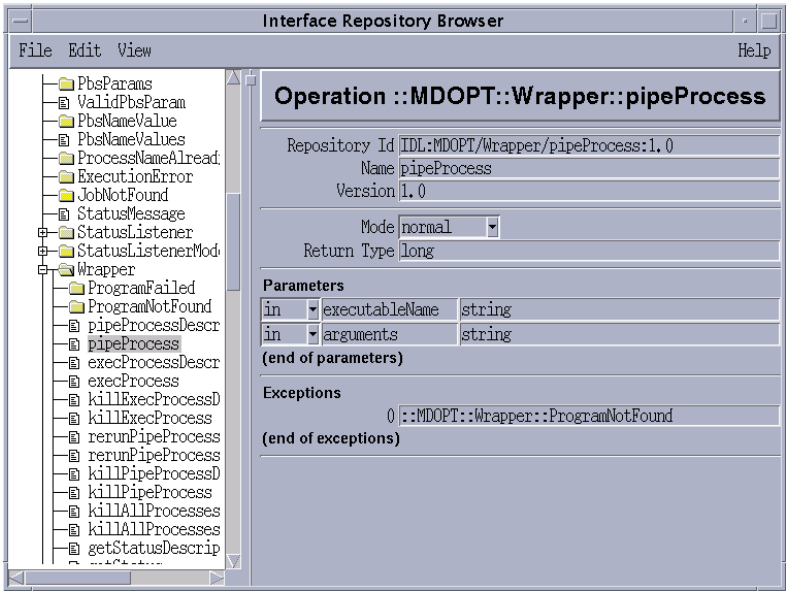

Figure 22. A Tcl IR browser client using Combat communicating with the C++ ORB MICO's IR

Appendix B – Programming a simple weather service client and server using Combat and Tcl

The Combat Tcl bindings make it very easy to create a CORBA client/server application. For instance, given an IDL definition for a simple weather service of:

```
struct Weather {
    double temperature;
    double humidity;
};
interface WeatherService {
    Weather getWeather(in string city, in string state);
    oneway void shutdown();
};
```

that defines a data structure called Weather that is returned by the getWeather method of the WeatherService interface, the corresponding Combat client written in Tcl would be:

```
eval corba::init \
    –ORBInitRef \
     NameService=corbaloc:iiop:myhost:9004/NameService  \
   -ORBInitRef \
     InterfaceRepository=corbaloc:iiop:myhost:9005/InterfaceRepository

corba::try {
    set ns [corba::resolve_initial_references NameService]
    set service [$ns resolve_str WeatherService]
    set output [$service getWeather “Austin” “Texas”]
    array set weather $output
    puts “temperature = $weather(temperature)”
    puts “humidity = $weather(humidity)”
} catch { ... e } {
    puts "Error: $e"
}
```

As shown, Combat maps IDL structs to a Tcl list of name value pairs. One benefit of this mapping is the weather service can return more information in the future and the Tcl code will continue to execute correctly.

A barebones implementation of the corresponding WeatherService server would look like:

```
 1 itcl::class WeatherService {
 2     inherit PortableServer::ServantBase
 3        private variable mypoa;
 4
 5     public method _Interface {} {
 6              return "IDL:WeatherService:1.0"
 7     }
 8     public method getWeather { city state } {
 9              return [list temperature 101.5 humidity 75]
10     }
11        public method shutdown {} {
12              puts "shutting down weather service"
13              $this destroy
14              puts "exiting"
15              global forever; set forever 1
16        }
17     public method destroy {} {
18              corba::try {
19                     set ns [corba::resolve_initial_references NameService]
20                     $ns unbind [list [list id WeatherService kind {}]]
21              } catch { ... e } {
22                     puts stderr "Error unregistering WeatherService: $e"
23              }
```

```
24              set obj [corba::resolve_initial_references POACurrent]
25              set poa [$obj get_POA]
26              set oid [$obj get_object_id]
27              $poa deactivate_object $oid
28              delete object $this
29      }
30         public method unregister {} {
31              corba::try {
32                     set ns [corba::resolve_initial_references NameService]
33                     $ns unbind [list [list id WeatherService kind {}]]
34              } catch { ...  e} {}
35         }
36         public method register {} {
37              $this unregister
38              set ns    [corba::resolve_initial_references NameService]
39              set ref [$this _this]
40           $ns bind  [list [list id "WeatherService" kind {}]] $ref
41         }
42         public method startWithDefaultPoa {} {
43              set mypoa   [corba::resolve_initial_references RootPOA]
44              set ref   [$this _this]
45              $this register
46         }
47         public method startWithCustomPoa {} {
48              set poa   [corba::resolve_initial_references RootPOA]
49              set mgr   [$poa the_POAManager]
50              set mypoa [$poa create_POA MyPOA $mgr \
51         {SINGLE_THREAD_MODEL RETAIN USER_ID IMPLICIT_ACTIVATION PERSISTENT}]
52              set oid   [$mypoa activate_object_with_id WeatherService $this]
53              set ref   [$mypoa id_to_reference WeatherService]
54              $this register
55         }
56 }
57
58 eval corba::init -ORBInitRef \
59         NameService=corbaloc:iiop:futuresparc1:9004/NameService \
60      -ORBInitRef \
61         InterfaceRepository=corbaloc:iiop:futuresparc1:9005/InterfaceRepository
62
63 #createobject
64 set server [WeatherService #auto]
65 #activate object
66 $server startWithCustomPoa
67 #$server startWithDefaultPoa
68 puts "server running"
69
70 set poa   [corba::resolve_initial_references RootPOA]
71 set mgr   [$poa the_POAManager]
72 #start accepting requests
73 $mgr activate
74 vwait forever
75 exit
```

Most of the non-implementation-specific incrTcl code for this server was generated using a Combat script that queries the interface repository for the necessary metadata and writes out the incrTcl classes based on a boilerplate template.

The CORBA ORB is initialized on line 58 with the locations of the naming service and the interface repository. An instance of the WeatherService object is created on line 64. At this point, the service is still not activated yet. For comparison, two different approaches are shown for activating the object. The “startWithDefaultPoa” method uses the implicit activation feature of the default portable object adaptor.

The "startWithCustomPoa" method creates a custom POA with some desired features such as single threaded dispatching and a user specified name so that the service will have the same name in it's IOR which when combined with a specified port number makes it easy to bind to the service even independently of a naming service. In both cases, the server registers itself with the naming service as shown in the register method on line 36.

The server starts receiving requests when POA manager is activated as shown on line 73. The "vwait" command shown on line 74 tells Tcl to enter into its event loop until "forever" changes its value.

Acknowledgements

MDOPT was developed under joint funding from the Air Force Research Laboratory, Wright-Patterson AFB, Ohio, under the Multi-Disciplinary Optimization Using Computational Fluid Dynamics, MDOPT, Contract Number F33615-98-2-3014, and from Boeing cost match funds. The work was performed by the Boeing Company, Phantom Works Organization, in Seattle. Dr. Don Kinsey and Lt Charles Hoke were the USAF Project Engineers. Gasper Fatta was the Boeing Program Manager and Dr. William Herling was the Principal Investigator. In addition to the authors, program development was completed by Mr. Dean Barron, Mr. Gordon Blom, Dr. Andrew Booker, Mr. Kwan Chang, Mr. Dr. Jonathan Elliot, Ranald Engelbeck, Dr. Paul Frank, Dr. Neal Mosbarger, Mr. David Treiber, and Dr. Matt Warfield. Contributions from Lt. Charles Hoke are gratefully acknowledged. His testing and feedback during the development of this system was a key component to the successful completion of MDOPT. In addition, acknowledgment is given to NASA for the technology contributions to several of the system modules. These include OVERFLOW, Chimera Grid Tools, CSCMDO, TWING, TLNS3D, HYPGEN and WINGDES. Finally, many thanks are given to Frank Pilhofer and Karel Gardas for their contributions to the Combat and MICO software and their conscientious support.

References

[1] Stephen T. LeDoux, William W. Herling, Joe Fatta, Robert R. Ratcliff, "Multidisciplinary Design Optimization System Using Higher Order Analysis Codes", 10th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference, 30 August - 1 September 2004, Albany, New York, AIAA 2004-4567.

[2] Tcl/Tk Developers website, URL: http://www.tcl.tk/ [cited August 2005]

[3] Combat website, URL: http://www.fpx.de/Combat/ [cited August 2005]

[4] CORBA 2.3.1 Specification, URL: http://www.omg.org/cgi-bin/doc?formal/99-10-07 [cited August 2005]

[5] MICO (**M**ICO **I**s **CO**RBA) website, URL: http://www.mico.org [cited August 2005]

[6] MySQL website, URL: http://www.mysql.com [cited May 2004]

[7] Portable Batch System, URL: http://www.openpbs.org, Altair Engineering, Inc, [cited August 2005]

[8] CORBA Product Matrix Comparison, URL: http://www.puder.org/corba/matrix [cited August 2005]

[9] MICO mailing list, "Announcement of MICO port to uCLinux", URL: http://www.mico.org/pipermail/mico-devel/2005-June/009334.html [cited August 2005]

[10] William W. Herling, Stephen T. LeDoux, Robert R. Ratcliff, "Application Studies using the 3DOPT Integrated Design System", AIAA Applied Multi-disciplinary Optimization Conference, AIAA-98-4720, September 1998.

[11] William W. Herling, Howard T. Emsley, Stephen T. LeDoux, Robert R. Ratcliff, David A. Treiber, Mathew J. Warfield, "3DOPT - An Integrated System for Aerodynamic Design Optimization", AIAA Paper 98-2514, June 1998, 16th Applied Aerodynamics Conference, Albuquerque, NM.

[12] Chad Smith, 2000, "[incr Tcl/Tk] from the Ground Up", McGraw-Hill.

[13] The TclJava development website, URL: http://tcljava.sourceforge.net/docs/website/index.html [cited August 2005]

[14] CORBA Event Service 1.1 Specification, URL: http://www.omg.org/technology/documents/formal/event_service.htm [cited August 2005]

[15] Tcom website, http://www.vex.net/~cthuang/tcom/ [cited August 2005]

[16] Java Desktop Components Website, https://jdnc.dev.java.net/ [cited August 2005]

[17] JaxFront Corporate Website, http://www.jaxfront.com/pages/ [cited August 2005]

[18] Object Management Group, "CORBA Reflection: OMG Request for Comments", OMG document mars/2004-08-12, 2004; URL: http://www.omg.org/cgi -bin/apps/doc?mars/04-08-12.pdf.

[19] Steve Vinoski, Douglas Schmidt, "XML Reflection for CORBA", C/C++ Users Journal, December 2003, URL: http://www.cuj.com/documents/s=8943/cujexp0312vinoski/ [cited August 2005]

[20] MySQLTcl (Tcl bindings for MySQL) Website, URL: http://www.xdobry.de/mysqltcl/ [cited August 2005]