Software Architecture of the Magdalena Ridge Observatory Inteferometer

Allen Farris[a], Dan Klinglesmith[a], John Seamons[a], Nicolas Torres[a], David Buscher[b], John Young[b]

[a] New Mexico Institute of Mining and Technology, Socorro, New Mexico, USA
[b] Cavendish Laboratory, University of Cambridge, Cambridge, UK

ABSTRACT

Merging software from 36 independent work packages into a coherent, unified software system with a lifespan of twenty years is the challenge faced by the Magdalena Ridge Observatory Interferometer (MROI). We solve this problem by using standardized interface software automatically generated from simple high-level descriptions of these systems, relying only on Linux, GNU, and POSIX without complex software such as CORBA. This approach, based on gigabit Ethernet with a TCP/IP protocol, provides the flexibility to integrate and manage diverse, independent systems using a centralized supervisory system that provides a database manager, data collectors, fault handling, and an operator interface.

Keywords: software design, control system, code generation, distributed system, interface protocol

1. MROI OPTICAL INTERFEROMETER

The Magdalena Ridge Observatory Interferometer[1] (MROI) is designed to be a 10-element 1.4 meter interferometer, in an equilateral "Y" configuration, operating in the visible and the near-infrared. It is being built on Magdalena Ridge, west of Socorro, New Mexico at an altitude of 3,200 meters. The Magdalena Ridge Observatory is a federally funded facility being built and managed by New Mexico Institute of Mining and Technology in conjunction with collaborators at the University of Cambridge (UK). It will be a companion to the fast-tracking 2.4 m telescope, currently in operation having obtained first light on Oct. 31, 2006.

The key science mission for the MROI is centered on three main areas: (1) studies of the environments of Active Galactic Nuclei (AGN); (2) stellar formation and the earliest phases of planet formation; and, (3) fundamental physics: stellar diameters, mass-loss, mass-transfer, convection and pulsation of single and multiple star systems.

The design goals for the MROI include: (1) being able to detect fringes on objects as faint as H=14, with a telescope aperture that provides diffraction limited images in the near-infrared using a low-order adaptive optics (tip-tilt) system; and, (2) providing model independent images, which implies that the interferometer should have a large number of telescopes and that the telescopes have the ability to be readily relocated onto a subset of 28 available telescope stations.

Major systems of MROI include: (1) supervisory system; (2) unit telescope, which includes the mount, optics, enclosure, wide field acquisition, and fast tip-tilt systems; (3) beam relay system; (4) vacuum system; (5) delay line system; (6) fringe tracker; (7) beam combiner system; (8) automated alignment system; and, (9) environmental monitoring system. References [2] – [7] contain additional information on these systems.

As a project MROI is broken into 36 work packages that are being designed and fabricated independently. Most of these have some computer control requirements. A significant fraction are being fabricated, and in some cases designed, by third parties. Our collaborators at the University of Cambridge function as overall system architects and are also responsible for implementing several of the key systems, including the delay lines and fast tip-tilt system.

The MROI beam combining facility was completed in February 2008. It includes all interconnected buildings on Magdalena Ridge housing the control facilities, optical, electrical and mechanical laboratories, delay lines, vacuum system, beam combiners and detectors. The MRO Interferometer is expected to see first light in 4th quarter of 2011, and obtain first fringes in 2012.

2. SOFTWARE DEVELOPMENT CHALLENGE

The essence of the challenge faced by the MROI software system can be easily stated: how do you merge independently developed systems into a coherent, robust, and unified software system that is intended to last for more than twenty years? The Supervisory System software must accomplish this goal; it must unify these systems in order for them to function as a single scientific instrument. Importantly, this goal must be accomplished without imposing unreasonable constraints on the systems; they must be left as independent as possible. Furthermore, the systems themselves must be able to be developed and tested as stand-alone systems without requiring an externally-imposed heavyweight software environment on them. This aspect of the MROI is significant because it allows the flexibility needed for future system expansion.

The systems that make up the MROI are implemented using very different techniques. For example, the telescope mount, designed and developed by AMOS (Liège, Belgium), is an internally complex system, with many low-level real-time components, controlled by LabVIEW. The telescope enclosure system, designed by EIE (Mestre, Italy), uses the BACnet protocol, commonly used in HVAC systems, to control environmental sensors, louvers, shutter, and rotation of the dome. This protocol is also used by the environmental control system in the MROI beam combining facility. The fringe tracker, designed and developed in-house by MRO, is built around FPGAs controlling lower-level devices and handling fast real-time data manipulations.

One traditional approach to solving this problem is to use CORBA-based middle-ware as the interface mechanism for communications between systems within a distributed processing context. This approach has been taken by ALMA[8], which uses the ACS (ALMA Common Software) system, and by the Keck Observatory, which uses a system called RTC (Real-time Control) that was developed by JPL. MROI has rejected this traditional approach because it imposes far too great a burden on the systems; it necessarily imposes a heavyweight software environment on them. Furthermore, CORBA is a very complex system that is extremely difficult to debug. It dominates the software architecture of applications that use it; it is more akin to an operating system than merely a communications mechanism.

The MROI has chosen instead to base its system software development on carefully minimizing its dependence on external software packages. We use Linux as our basic operating system and GCC (GNU Compiler Collection), together with the Java and C programming languages. We also use a standard relational database (PostgreSQL) and a real-time add-on to Linux (Xenomai). These are the tools that are used to craft and operate the Supervisory System and its interface to the independently developed systems. We have developed our own communications protocol within this context, based on gigabit Ethernet using TCP/IP. Carefully choosing these dependencies will enable us to maintain this software system over its twenty-plus year lifespan.

3. SOFTWARE DESIGN

3.1 The Supervisory System

The MROI Supervisory System is the high-level software system that manages the entire MRO Interferometer. It is responsible for starting, stopping, and configuring the entire system. It has access to a monitor and configuration database that houses all data needed to configure and start the system, as well as storing data accumulated during the execution of the system. It has a high-level interface to each active system that makes up the MRO Interferometer, allowing it to start, stop, monitor and control all aspects of the software and hardware. The actions of the Supervisory System take place regardless of whether the system is executing science observations, engaged in system-level calibrations, diagnostic testing, or merely idle.

The MROI Supervisory System consists of the following software modules:

- Executive – Start, stop, and monitor the status of all active systems, including the Supervisory System.
- Operator Interface – The mechanism by which the telescope operator interacts with the Supervisory System.
- Supervisor – Manage one or more unit telescopes to support scientific observations, system calibrations, or diagnostic testing.

- Fault Manager – Determine the significance of all faults and take appropriate action to handle them.
- Data Collector – Collect all software logs, engineering monitor data and science data from all active systems.
- Database Manager – Manage all access to the permanent database, including accessing and updating configuration data, as well as reformatting data in the data collector to be permanently stored in the database.

There is one Executive and Database Manager in the system. Likewise, there is only one Operator Interface, even though this module may interact with more than one operator workstation. While the Data Collector functions as a unified system, there will be many Data Collection instances collecting data from different systems and possibly running on different computers. There must be at least one Supervisor in the system but there may be more than one. There is one Fault Manager for each Supervisor.

The Executive is responsible for starting the entire system, including all systems and the Supervisory System itself. The Executive uses a pre-defined start-up scenario to determine which systems to start and their configuration. For example, the Executive may start more than one Supervisor, i.e. the array of unit telescopes may be divided into independently operating sub-arrays. The Executive constantly monitors systems and computers for failures by providing a watchdog service that periodically interrogates each active system to see if it is still functional and to discover what state it is in. The software interface to each system implements a well-defined state model. The normal sequence of states is: start, initialize, operational, shutdown, and stop. It not only communicates any failures to the telescope operator, it also informs relevant Supervisors and Fault Managers and leaves it to them to determine the significance of the failure. Systems can be re-initialized and placed back into service.

The purpose of the Operator Interface is to define the relationship between the telescope operator and the Supervisory System in general. This module defines the mechanism by which the telescope operator controls the system and also the mechanism by which systems may communicate with the telescope operator. Such commands may include: telling a specified supervisor to begin executing a particular science project, terminating the current executions by a particular supervisor, reconfiguring the current array of unit telescopes, or taking a particular unit telescope offline or placing it in a diagnostic mode. There may be more than one person monitoring and controlling different supervisors and sub-arrays. Nevertheless, this Operator Interface handles the problem of dealing with these multiple workstations; the rest of the Supervisory System and the systems see a single telescope operator interface, regardless of how the associated data is distributed to various workstations. Another important point to make is that systems can send messages directly to the telescope operator; the design of the Operator Interface does not require the routing of traffic through a single process.

The major purpose of the Supervisor is to carry out meaningful work on the telescope. This may include executing science observations or diagnostic tests. The Supervisor manages a particular set of resources assigned to it by the Executive. This may include a set of unit telescopes, beam relays, delay lines, fringe tracker and beam combiner. The first thing the Supervisor does is to create a Fault Manager that handles all faults from the resources that it manages. It then creates an interface to all its systems, informing them of the Fault Manager to which to send faults and alerts.

There are two types of Supervisors – one that is highly structured and another that is very interactive. When the Executive creates a Supervisor, it creates the Supervisor in one of these two modes. These two modes are used in the following manner: (1) executing structured science observations, prepared under the control of the Observation Preparation Tool, and (2) executing diagnostic testing under the interactive control of a staff member. The essential difference between these is that the first is highly structured, while the second is not. System level calibrations, which may be necessary before executing a series of science observations, are regarded a falling into the first category, because they are pre-defined and structured.

A Fault Manager handles faults and alerts that are generated by the resources managed by a particular Supervisor. A fault is a detected malfunction within a hardware or software component in a system, including detecting some monitored quantity that falls outside some specified range. A system is the collection of hardware and software components online at any time. An alert is a reference to a condition requiring the immediate attention of an intelligent agent lying outside the current system. When a software system detects a fault, it may or may not also issue an alert; it depends on the significance of the fault. However, in almost all cases, an alert will have a fault associated with it. The Fault Manager module contains an Alert management system in addition to handling faults. This system monitors alerts

and maintains their status until action is taken to clear them. Any system can send an Alert (plus the fault that caused the alert) directly to the telescope operator.

The Fault Manager receives a message identifying a fault, evaluates the significance of that fault, and takes some action with regard to that fault. It is a rule-based system. Understanding the significance of a fault is highly context dependent. Among other things it depends on the current activity in the array; a fault may be insignificant if the array is idle, but may be extremely significant if the array is engaged in science observations. In addition, such an evaluation requires a good deal of information about the fault itself. It is rarely the case that the component initially identifying the fault has sufficient contextual information to evaluate the significance of a fault.

There are many Data Collectors within the Supervisory System, configured and deployed by the Executive. These data collectors nevertheless function as a single system. A particular Data Collector collects data from one or more specific system instances. All Data Collector instances are generic, i.e. they have the same structure; they merely differ in how they are configured. A Data Collector object may reside on the same computer that generates most of the data or on one close by; but it could reside anywhere on the network. The configuration data, obtained from the database by the Executive, tells the Data Collector how to connect to specific systems and what kind of data to collect. A Data Collector has no knowledge of the internal structure of the applications or significance of the data; it merely collects data. It does contain meta-data that describe the attributes of the data being collected because these attributes are obtained from the database. The Data Collector system is intended to be simple and to rely on generic properties of the data and systems with which it interacts.

A Data Collector object has a thread for each system instance from which it collects data. That thread listens for data that are published on a particular port belonging to that system and stores the collected data in a buffer. There is another thread that does any necessary reformatting of the data in order to periodically store the collected data in the database. In addition to these activities the Data Collector has a publish/subscribe thread that allows any client to register with the Data Collector system in order to receive monitored data in real time.

The publish/subscribe system is a special feature of the Data Collector system. An external client may elect to receive monitored data collected in real time. The external client interacts with the Executive, because the Executive keeps track of how particular Data Collectors are deployed and what they are monitoring. The client specifies a computer and port on which to receive the data, the name of the system and monitor point about which to receive data, and the frequency with which to receive the data. The Executive then registers the client with the appropriate Data Collector. After these actions are performed, the Data Collector sends the requested data directly to the client at the requested frequency.

The Database Manager controls access and update to the monitor and configuration database. The monitor and configuration database is implemented as a relational database. The specific properties as a relational database are well-hidden by a high-level interface implemented by the Database Manager. There is one Database Manager in the Supervisory System. It is a Java-based system that depends on the standard JDBC interface to relational databases. This module encapsulates the dependence on a relational database; the rest of the system only depends on the high-level interface provided by the Database Manager, not on the details of database structure or SQL statements. This database provides an integrated view of the total system and is intended to house all data needed to operator the telescope and store all data that are significant to its operation, including monitor and engineering data, system logs, and data collected during science observations.

The monitor and configuration database is a crucial portion of the Supervisory System. The database contains one or more system configurations. A configuration is a specification of active computers and systems. There is one special configuration, designated as the startup configuration, which is used to start the system. The database tracks changes in the system configuration over time and also tracks when a particular configuration is started and stopped. All configurations over time are permanently recorded in the database. There is very little overhead in doing this, since only references to computers and systems are stored in the configuration table. In this manner one can always tell which telescope configuration generated particular system logs, monitor data, and science data.

3.2 General Structure of the MROI Software System

The general structure of the MROI software is depicted in Figure 1, which is a simplified depiction of its major systems.

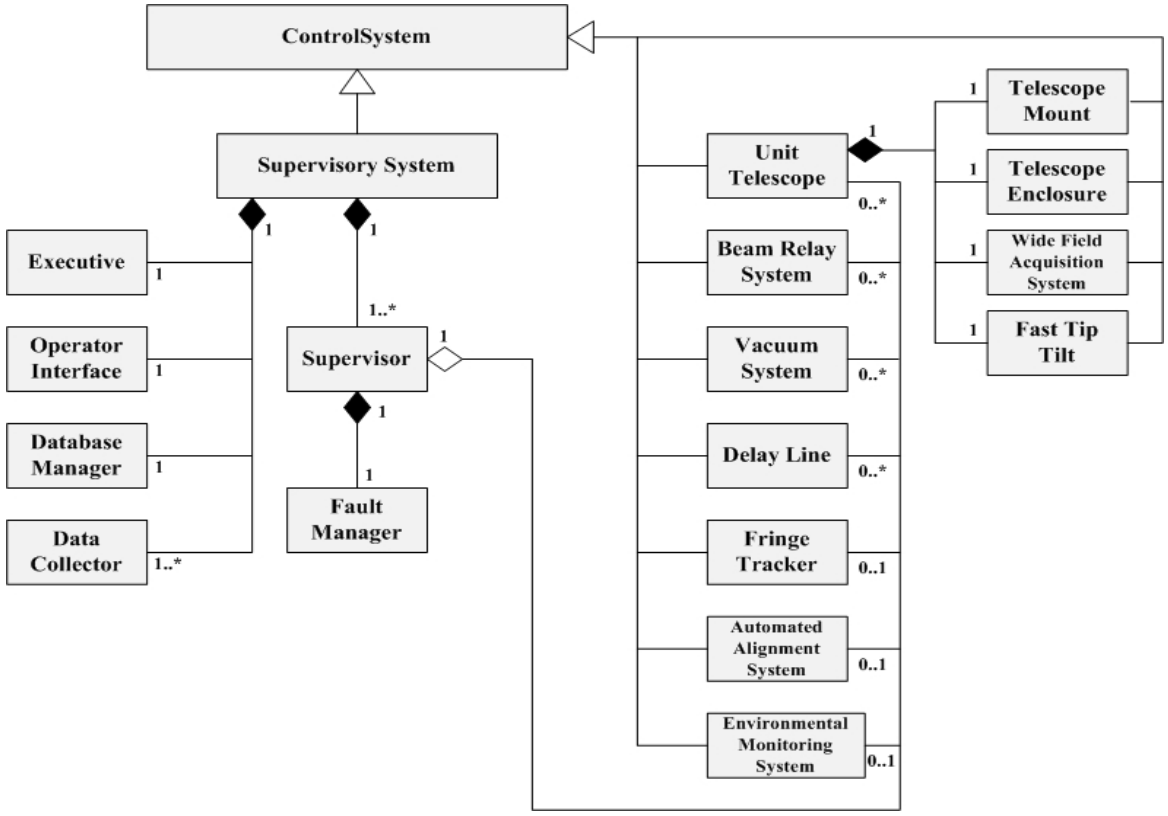

Figure 1. Structure of the MROI software system in UML notation

The features common to any system are encapsulated in the ControlSystem module. In general, any system instance contains the following features:

- Has a specific name that is unique across the entire network,
- Is of a specific type,
- Has a state at all times – it implements a well-defined state model,
- Has a log file, on which it publishes messages related to internal events that occur within its lifetime,
- Publishes monitor data which is produced during its execution,
- Is able to communicate with other remote servers, including, but not necessarily limited to, the Database Manager, Fault Manager, and Telescope Operator,
- Functions as a server, i.e., it responds to commands sent by one or more remote clients, but, more specifically, it functions under the control of a Supervisor.

All of this functionality is implemented by a suite of classes that are used in defining an MROI application system. Using its connection to the Database Manager, a system is able to ask for initialization data. Similarly, using its connection to the Fault Manager, it publishes faults and alerts, and, if necessary, can send messages directly to the Telescope Operator. Each system is an extension of the ControlSystem module, which implements this basic functionality. In addition, each system has a high-level interface that is used by the Supervisor to command and control it.

3.3 High-level Definition of a System

The definition of the high-level interface to a system is expressed using a spreadsheet, actually a set of spreadsheets. This definition also imposes requirements on the system to implement the functionality expressed in the spreadsheets. Two spreadsheet applications have been used and tested to create the high-level interface definition of a system: Microsoft's Excel and OpenOffice's Calc. These spreadsheets are then input to a code generation framework that generates the required interface code.

There are five worksheets contained in the spreadsheet, which are described in the following sections.

3.3.1 System worksheet

The System worksheet defines attributes of the system as a whole, including the formal document that serves as the primary description of the system. It is understood that the system conforms to and is an implementation of the requirements and concepts in this interface protocol.

There can be more than one system described in a given spreadsheet. For example, a given system, such as the Environmental Monitoring System, may consist of a collection of different types of systems: weather station, all-sky camera, seeing monitor, dust monitor, etc. These can all be described in a single spreadsheet with the subordinate systems being a part of the overall Environmental Monitoring System.

The column names and their meaning are:

- Name – The short name by which this type of system is known.
- Description – A brief description of the system.
- Package – The package name of this system (mainly for Java systems).
- Import – Additional files (a series of names separated by spaces) to be imported for the system.
- Full Name – The full name of this system.
- Extends – The name of a system of which this one is an extension.
- Parent System – The name of the system to which this system belongs. This entry allows the definition of multiple systems that are contained within a master system.
- Implement – Whether this system is to be implemented or not; and, if so, whether it is a Java or C system.
- Is Asynchronous – Does this system implement asynchronous methods?
- Is A Monitor – Does this system produce monitor data?
- Work Package – The MROI work package to which this system belongs.
- Document Title – The title of the document that serves as the primary description of this system.
- Document Number – The MROI document number of the primary document.
- Document Issue – The revision number of the primary document to which this system conforms.
- Document Date – The date of the primary document to which this system conforms.

3.3.2 Monitor worksheet

The Monitor worksheet defines monitor points, which includes engineering data routinely produced to describe internal conditions and science data, such as images, that result from executing specific commands. Names of monitor points must be unique within the system to which they belong.

The column names and their meaning are:

- Name – The name of this monitor point.
- System – The name of the system (from the System worksheet) to which this monitor point belongs.
- Description – A brief description of this monitor point.
- Returns – The data type that this monitor point returns.
- Can Be Null – Can this monitor point return a null value?
- Throws Exception – Can this monitor point return an exception?

- Asynchronous – Is this monitor points implemented by an asynchronous method?
- Data Unit – The physical unit that describes this monitor point (this is the unit that is used in archiving this data), called the canonical value.
- Minimum Value – The maximum value of this monitor point.
- Maximum Value – The minimum value of this monitor point.
- Default Value – A default for this monitor point (used in simulations).
- System Unit – The physical unit that is used internally in this system in describing a raw value of this monitor point.
- Raw Data Type – The data type associated with the raw value of this monitor point.
- Scale – The scale used to convert a raw value to a canonical value. The formula is canonical-value = raw-value * scale + offset.
- Offset – The offset used to convert a raw value to a canonical value.
- Mode – The system state in which this monitor point can be executed, usually 'any'. For example, if labeled `diagnostic' then this monitor point can only be executed in diagnostic mode.
- Implement – Should a method be generated to execute this monitor point?
- Archive Interval (secs) – The interval, in seconds, at which this monitor point should be stored in the archive.
- Archive Only On Change – Should this monitor point be archived only when it changes in value?
- Display Unit – The units in which to display, in a GUI, values retrieved from the archive for this monitor point.
- Graph Minimum – The graph minimum to be used in a GUI display.
- Graph Maximum – The graph maximum to be used in a GUI display.
- Graph Title – The title to be used in a GUI display.

3.3.3 Fault worksheet

The Fault worksheet describes each fault that might be generated by the system. Names of faults must be unique within the system to which they belong.

The column names and their meaning are:

- Fault Name – The name of this fault condition.
- System – The name of the system (from the System worksheet) to which this fault belongs.
- Monitor Point – The name of the monitor point, if any, (from the Monitor worksheet) to which this fault belongs. A fault condition may be associated with the system as whole, in which case `none' should be entered.
- Description – A brief description of the fault condition.
- Fault Condition – An expression that triggers the fault condition.
- Fault Severity – The level of severity associated with this fault.
- Fault Action – A list of named actions to be taken if the fault condition arises.

3.3.4 Control worksheet

The Control worksheet defines commands used to initiate actions within the system. It describes each command in the system. Parameters that are associated with these commands are in the Parameters worksheet. Names of commands must be unique within the system to which they belong.

The column names and their meaning are:

- Name – The name of this command.
- System – The name of the system (from the System worksheet) to which this command belongs.
- Description – A brief description of this command.
- Returns – The data type that this command returns.
- Can Be Null – Can this command return a null value?
- Throws Exception – Can this command return an exception?
- Asynchronous – Is this command implemented by an asynchronous method?

- Mode – The system state in which this command can be executed, usually 'any'. For example, if labeled 'diagnostic' then this command can only be executed in diagnostic mode.
- Implement – Should a method be generated to execute this command?

3.3.5 Parameters worksheet

The Parameters worksheet defines the parameters that are associated with specific commands or with the system as a whole. Parameter names associated with commands only have to be unique within the context of the command to which they belong. Names of parameters that belong to the system as a whole are required to be unique within that entire context.

The column names and their meaning are:

- Parameter Name – The name of this parameter.
- System – The name of the system (from the System worksheet) to which this parameter belongs.
- Command – The name of the control command (from the Control worksheet) to which this parameter belongs, if any. If this parameter belongs to the system as a whole, then `none' should be entered.
- Description – A brief description of this command.
- Required – Is this parameter required?
- Data Type – The data type of this parameter.
- Data Unit – The physical unit that describes this parameter. This is the canonical value, the unit used by an external agent in sending data to this system.
- Minimum Value – The maximum value of this parameter, in canonical units.
- Maximum Value – The minimum value of this parameter, in canonical units.
- Default Value – A default for this parameter (used in simulations), in canonical units.
- System Unit – The physical unit that is used internally in this system in describing a raw value of this parameter.
- Raw Data Type – The data type associated with the raw value of this parameter.
- Scale – The scale used to convert a raw value to a canonical value. The formula is canonical-value = raw-value * scale + offset.
- Offset – The offset used to convert a raw value to a canonical value.

3.3.6 Code Generation

The worksheets are the input to a code generation framework, which generates: (1) client/server interfaces to the system, using a communications protocol based on TCP/IP, (2) database entries describing all system monitor points and commands, (3) code necessary to publish and access real-time monitor data streams, convert data to database format, and insert data into the database, (4) test programs - to test basic features of monitor points and commands, (5) generic engineering GUIs, and (6) base classes to form a full simulation interface.

The code generation framework is based on a language called Xpand. Xpand began in early 2000s as part of a more general modeling framework by OpenArchitectureWare, an association of software developers in Germany and associated with several German companies. Xpand is a mark-up language for defining templates to generate text and was a key part of the modeling framework. This modeling framework has since been modified considerably and incorporated into a very complex Eclipse project: the Eclipse Modeling Framework (EMF), described at http://www.eclipse.org. At MROI we have taken the original Xpand language (slightly modified) and implemented the code generator as a stand-alone process, independent of EMF. As such it is only dependent on standard Java.

The code generation process is based on a model and templates. We emphasize code but it could be any kind of text. The model is defined and implemented using Java. Templates are defined using a simple, text-based, mark-up language. This language has “define” modules (functions), “expand” statements (function calls), looping statements and if-statement constructs, as well as local variables. It also has simple text statements containing expressions that reference attributes of the model. Together with “file” statements, these text statements are written to the actual file that is being generated.

Code generation consists of: loading the templates, creating the model by instantiating the Java classes, and generating the text files using the templates together with the instantiated model. The templates contain expressions that reference attributes of the model objects.

The model is defined by the application that uses the Xpand framework. There are no constraints on the complexity of the model or on how it is defined and created. The only conditions on the model are that it must be a set of Java classes and have some method to create an actual instance of the model. The Xpand processor uses Java reflection extensively to capture attributes of the instantiated model. Unlike some code generation frameworks with arcane syntax, the Xpand templates are very readable and, together with the model, are quite natural from a Java perspective.

3.4 Interface Protocol

The interface protocol is based on TCP/IP. The protocol implements a basic method call:

RtnType method (Type1 parm1, Type2 parm2, ...) throws AnException;

Supported data types include Boolean, byte, integers (16-bit, 32-bit, 64-bit) , float, double, UTF-8 encoded character strings, enumerations, and pre-defined structures, as well as one-dimensional arrays of these basic types. The protocol handles byte-swapping. Java and C are supported; other languages could be added, if necessary. A set of internal functions encodes a client request into a byte stream and a similar set of functions decodes this byte stream on the server side into an appropriate method call.

Both synchronous and asynchronous commands are supported. These are defined from the client's perspective. With a synchronous command the client blocks until the command has executed. With an asynchronous command, the client does not block, but the results of the execution are delivered to the client at a later time.

The structure of a system in indicated in Figure 2, which depicts a WeatherStation. The software in is three layers: basic classes that are common to all systems, a middle layer that is automatically generated from the spreadsheets that define the high-level interface, and the actual implementation of the WeatherStation itself. Both client and server classes are present; the WeatherStationClient contains all the code that is necessary for any client to communicate and control a WeatherStation.

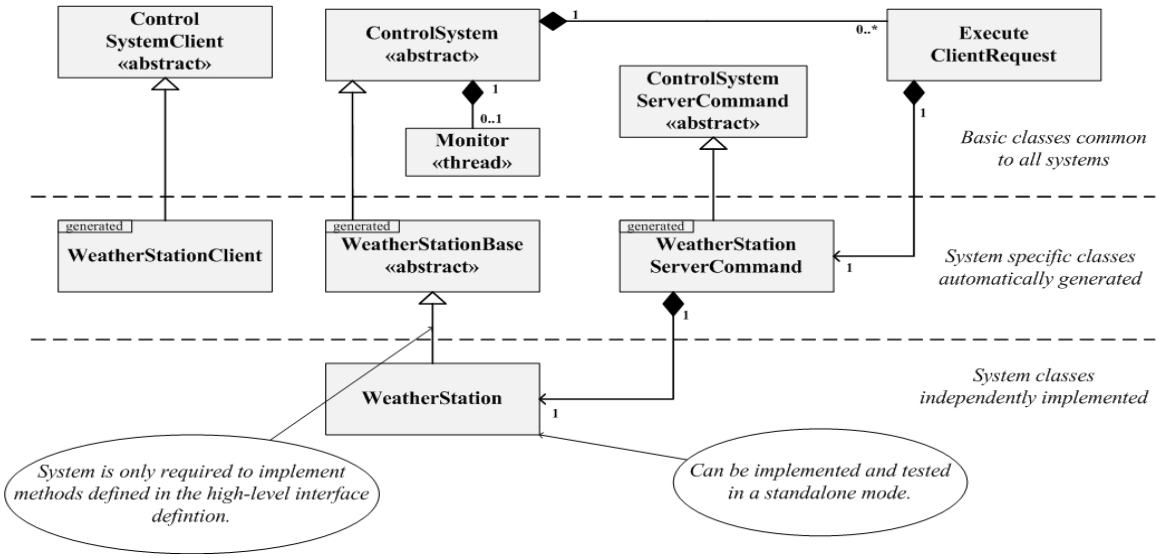

Figure 2. Implementing a control system

It is important to realize that the WeatherStation itself can be developed and tested without using the complexity of the client-server protocol. All that is required is for the WeatherStation to implement all of the methods that it described in the spreadsheets. In fact we have plans to develop a GUI-based, stand-alone version of the Supervisory System, which can be used to operate, test and debug a system in a completely standalone mode, without using any network facilities.

4. COMPARISON TO CORBA-BASED SYSTEMS

There are four key areas that any distributed processing system must accommodate. It is instructive to compare the MROI approach to the CORBA-based approach in each of these areas.

(1) Any distributed system must have some mechanism for managing basic system components, i.e. starting and stopping objects representing systems. CORBA uses its Naming Service and Request Invocation to manage references to objects. MROI uses the Executive together with the system configuration definition in the database to start and monitor high-level system objects.

(2) Any distributed system must use some interface protocol. Within CORBA this is accomplished with its Interface Definition Language (IDL) together with the defined language bindings and software tools that compile the IDL statements into specific language source code. MROI uses a code generation process to generate its TCP/IP interface protocol.

(3) Most distributed systems require some form of dynamic object discovery; i.e. the system must allow for changes in the dynamic configuration of its components, taking them off-line and putting them on-line, etc. Within CORBA, this feature is a source of much of its complexity and is accomplished using its Trading Service. Within MROI the Executive manages system resource reallocation and the database maintains and tracks all system configuration changes.

(4) Most distributed systems must manage asynchronous events, i.e. events that occur at unpredictable times that are important to the operation and status of the system. CORBA uses its Event Service, which is a decoupled communications model. Within MROI this aspect of the system is accomplished with the Fault Manager and the Data Collectors which have a publish/subscribe mechanism for distributing data to any client.

In all cases, the MROI approach is implemented far more naturally within the context of the system architecture and without introducing the complexity required by CORBA. Moreover, in cases 1, 3, and 4, any application using CORBA must implement at least some of the work involved in these tasks. In case 2, the interface protocol, the code generation process is used in MROI, which is not nearly as difficult to implement as implementing an IDL language compiler. Not only does one have far more control over the generated code, one also gets the added benefit of generating additional products, such as database definitions, GUIs, test programs, and simulations.

5. CURRENT STATUS

At present the Supervisory System has been completely designed and partially implemented. Code generation is being used extensively and the database design and database interface software have been implemented. Basic supervisory software is working, but we have only rudimentary versions of the Fault Manager, Data Collector, and the Operator Interface. Comprehensive GUIs are not yet implemented. The system interface definition process has been implemented and tested for Java systems. The C system interface design is complete and partially implemented.

Since the approach defined here is a completely general client-server protocol, we have used it to implement a project document repository with Kerberos security added onto the interface protocol. There was a general dissatisfaction with the current repository, so it was not particularly difficult to define a new database built around our concept of work packages that uses the system definition and code generation process. This system is currently being tested. Adding Kerberos as an optional security protocol will, in the future, allow us to define a client that can remotely, i.e. from outside the firewall, interact with the MROI system from anywhere on the Internet and do so in a secure manner.

By fall of 2010, we will implement the Supervisory System in a comprehensive simulation environment that will simulate what is needed for a Unit Telescope, including the Telescope Mount, Wide Angle Sensor, and Environmental

Monitoring systems. Other systems, including the Enclosure and Fast Tip-Tilt systems, will be added later. Our top priority is to support site acceptance testing and commissioning activities for the first unit telescope scheduled for 4th quarter 2011. Various features will continue to be developed as the second telescope is added and additional systems, such as the automated alignment system, are added to the overall software framework.

ACKNOWLEDGMENTS

The Magdalena Ridge Observatory is funded by Agreement No. N00173-01-2-C902 with the Naval Research Laboratory (NRL). MROI is hosted by the New Mexico Institute of Mining and Technology (NMT) at Socorro, NM, USA, in collaboration with the University of Cambridge (UK). Our collaborators at the University of Cambridge wish to also acknowledge their funding via Science and Technology Facilities Council (STFC) in the UK.

REFERENCES

[1] Michelle J. Creech-Eakman, et al., "Magdalena Ridge Observatory interferometer: advancing to first light and new science", Proc. SPIE 7734, paper 7734-5 (2010)

[2] Olivier Pirnay, "Magdalena Ridge interferometer: assembly, integration, and testing of the unit telescopes", Proc SPIE 7734, paper 7734-38 (2010)

[3] Ifan Payne, "Innovative enclosure design for the MROI array telescopes", Proc SPIE 7739, paper 7739-144 (2010)

[4] Fernando Santoro, et al., "Mechanical design of the Magdalena Ridge Observatory interferometer", Proc SPIE 7739, paper 7739-151 (2010)

[5] Martin Fisher, et al., "Design of the MROI delay line optical path compensator", Proc SPIE 7734, paper 7734-156 (2010)

[6] Colby A. Jurgenson, et al., "The MROI fringe tracker: laboratory fringes and progress toward first light", Proc SPIE 7734 paper 7734-153 (2010)

[7] Alisa V. Shtromberg, et al., "Magdalena Ridge Observatory interferometer automated alignment system", Proc SPIE 7734, paper 7734-39 (2010)

[8] J. Schwarz, et al., "The ALMA Common Software – Dispatch from the trenches", Proc SPIE 7019, paper 7019-32 (2008)