Astronomical Data Analysis Software and Systems X
ASP Conference Series, Vol. 238, 2001
F. R. Harnden Jr., F. A. Primini, and H. E. Payne, eds.

SOFIA's CORBA Experiences: Instances of Software Development

John Graybeal[1], Robert Krzaczek[2], John Milburn[3]

Stratospheric Observatory for Infrared Astronomy, NASA Ames Research Center, MS 207-1, Moffett Field, CA 94035

Abstract. Developing data systems for special purpose applications—like one-of-a-kind telescopes—is a singular, if not idiosyncratic, process. Developers must master and wisely use rapidly changing software technologies to produce systems faster, better, and cheaper, meanwhile keeping up with iterative requirements and schedules. Architectural standards such as CORBA may help—or may lead to slow, hard to change, and expensive data systems.

The Stratospheric Observatory for Infrared Astronomy (SOFIA) will use CORBA in several different environments—the airborne mission systems (MCS), the ground support system (DCS), and a Facility Science Instrument (FLITECAM). A review of CORBA development experiences on the MCS reflects the challenges and choices made, while comparison with other SOFIA implementations shows the variety of CORBA applications and benefits.

1. Introduction

1.1. The Stratospheric Observatory for Infrared Astronomy

The Stratospheric Observatory for Infrared Astronomy (SOFIA)[4] is a major infrared and submillimeter observatory scheduled to begin operations within two years. A joint collaboration of NASA and DLR (the German Air and Space Agency), the Boeing 747-SP aircraft will carry the 2.5-meter telescope at or above 12.5 km, where the telescope will collect radiation primarily in the wavelength range from 0.3 micrometers to 1.6 millimeters. With a 20-year operational lifetime, SOFIA is designed to maximize astronomical value per unit cost, as compared to the other observing platforms such as balloons or satellites. To meet this goal, it must be a particularly flexible, long lasting and economic platform for conducting research. The SOFIA data systems will be key to achieving these objectives.

---

[1]Logicon Sterling Federal

[2]Rochester Institute of Technology, Center for Imaging Studies

[3]University of California, Los Angeles, Astronomy Department

[4]`http://sofia.arc.nasa.gov/`

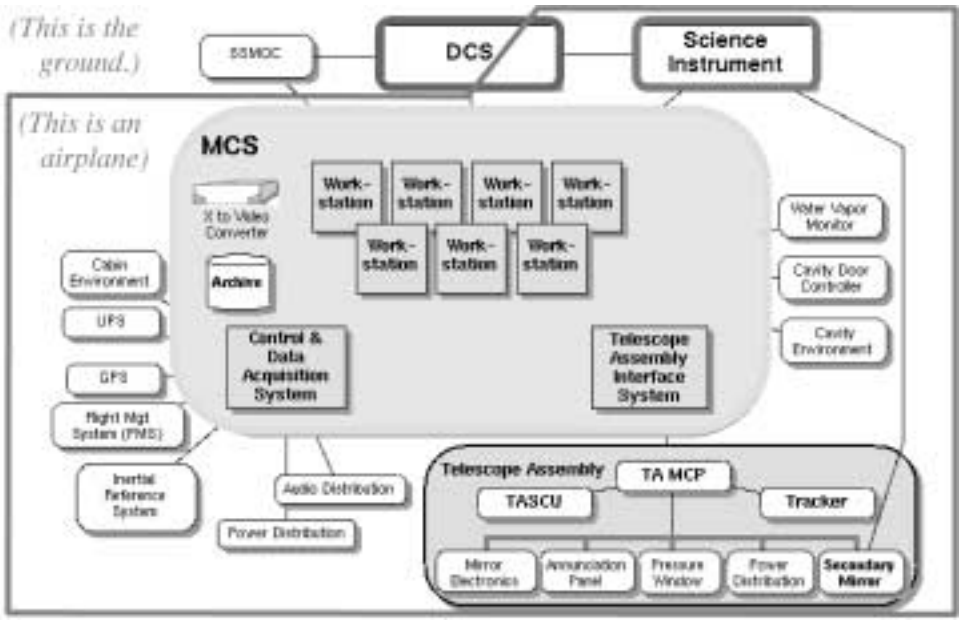

Figure 1. SOFIA's Software Components

1.2. SOFIA Software Overview

SOFIA has five principal software development environments:

- Science Instruments (SIs),
- the Mission Control Subsystem (MCS),
- the Telescope Assembly (TA),
- the Data Cycle System (DCS) and
- custom dedicated subsystems (for example, the water vapor monitor).

Different organizations develop software components in each environment, and the subsystem interfaces must be carefully negotiated and defined. As might be expected, functional requirements differ for each of the components, and development standards differ for the various environments. These variations all affect whether, and how, CORBA is used in SOFIA software.

In the following subsections we describe each of SOFIA's software components, so that their use in SOFIA—and CORBA's use in each component—will be clearer. Figure 1 provides graphical context for these descriptions.

1.3. Science Instruments

SOFIA's numerous Science Instruments, or SIs, are developed by science teams according to specifications provided by Universities Space Research Association (USRA), the prime U.S. contractor for SOFIA. The observatory will change SIs as often as every week, thereby supporting a wide variety of scientific investigations over its 20-year life cycle, while using the latest science instrumentation.

There are two classes of science instruments, Facility Science Instruments (FSIs) and Principal Investigator Science Instruments. FSIs will be operated by the observatory, and as such must undergo more thorough review and meet

more challenging development criteria. The First Light Infrared Test Experiment Camera (FLITECAM) is an FSI designed to perform critical checkout and integration phase activities for SOFIA.

On SOFIA, science instruments control the observatory, but only partially. Scientists using SOFIA will command science operations through an SI's user interface, and the SI in turn commands some aspects of the observatory through the MCS. But other observatory controls must be performed by observatory staff using their own interfaces. (The Data Cycle System, described below, will eventually provide a consistent interface for many of the science instruments, and perhaps even for the observatory itself.)

1.4. The Mission Control Subsystem

The Mission Control Subsystem (MCS) is the command and control heart of SOFIA. It communicates with the observatory subsystems (including the TA), coordinates their actions, and provides an interface for science instruments, scientists, observatory staff, and other users to interact with the observatory. To make this interface as accessible as possible, the MCS implements an ASCII-based command language, available over a standard TCP/IP socket interface.

The MCS is a highly distributed system with a high-speed network of nine workstations, most running the Solaris operating system. The MCS must achieve reliable, high throughput for both SOFIA mission housekeeping data and science commands. (Science data does not go through the MCS but is maintained internally by all SIs and also by the DCS for the FSIs.) The MCS must provide enough configurability and flexibility to serve as the SOFIA baseline data system for 20 years of science operations (Papke et al. 2000).

1.5. The Telescope Assembly

The Telescope Assembly (TA) performs control and pointing for SOFIA's telescope and related components. It is being developed by a consortium of companies under the direction of the Deutschen Zentrum für Luft- und Raumfahrt e.V. (DLR, the German space agency). Although the TA's subsystems communicate with each other to some degree, the MCS is responsible for coordinating their work to perform science effectively.

1.6. The Data Cycle System

The Data Cycle System (DCS) provides an observatory-level, science-oriented interface to SOFIA. On the ground it facilitates science interactions for all parts of the SOFIA data life cycle. During flights the DCS standardizes science functions available through the SOFIA science instruments. For both of these functions, the DCS must be capable of rapid reconfiguration to address a wide variety of science functions and interfaces.

The first DCS implementation will provide basic functions for the SOFIA Facility Science Instruments. The DCS will eventually support all science instruments that take advantage of it.

1.7. Custom Dedicated Subsystems

SOFIA's custom dedicated subsystems include the water vapor monitor, the Cavity Door Control Subsystem, the Environment Control Subsystem, and the Mission Audio Distribution System. These are dedicated systems performing specialized functions in support of the SOFIA mission. Most software is implemented within dedicated embedded systems, and if a subsystem communicates with the MCS, it usually uses a subset of the SOFIA Command Language.

2. Key CORBA Attributes

Briefly, for those unfamiliar with CORBA (Common Object Request Broker Architecture), it is a standard which describes certain "middleware" products. These products provide an infrastructure on top of which application features may be developed. CORBA specifies a certain set of features useful in an object-oriented, distributed, multi-processor environment, including:

- object registration, location, and activation,
- naming and trading services (find objects based on their names and properties),
- parameter marshaling and De Marshalling (send data between systems),
- event notification and
- object life cycle management, security, and transactions.

In summary, CORBA's features generalize typical object-oriented behaviors (access, method calls, etc.) from a single-processor environment to a distributed environment.

One feature not built into CORBA is low-latency and real-time distribution of data. Systems with real-time constraints would typically be designed to avoid the need for data marshaling services that help convert data objects to different language and operating system formats in a distributed system. Such systems typically use consistent languages and operating systems.

3. SOFIA Environments Not Using CORBA

Some of the SOFIA software architectures have compelling rationales for not using CORBA. The Telescope Assembly and custom dedicated subsystems do not incorporate CORBA at all. Those subsystems' software is analogous to firmware—largely stable code with specific functionality, typically implemented on single embedded CPUs with limited interface complexity. Given stable system requirements, pre-established stand-alone development environments and no need for on-the-fly component associations, CORBA would add little value.

Many of the science instruments have environments similar to the "embedded subsystems" just described. They are also fairly stable systems, with unchanging connectivity needs, and often use only one or few processors and a single programming language and operating system. While the functional complexity of many of these science instruments (and other non-CORBA systems) is quite high, functional complexity alone does not imply the need for CORBA. For most science instruments, it is more appropriate to build the software for

the expected needs, designing only the specific flexibility that has been defined, and then make changes in the developed software on an as-needed basis.

4. SOFIA Products Using CORBA

By reviewing the three SOFIA software products that do use CORBA—DCS, MCS, and the FLITECAM FSI—we identified a set of attributes that contributed to the decision to use CORBA. Not surprisingly, these attributes (see Table 1) are well matched to CORBA's intended environment.

Table 1. CORBA Motivators

Factor	DCS[a]	MCS[a]	FLITECAM[a]
Number of Operating System Types	●●●	●●	●●
Number of Processors (i.e., CPUs)	●●●	●●●	●●
Number of Supported Languages	●●●	●	●●
Object Orientation of the System	●●	●●	●●●
Amount of Data Distribution	●●●	●●	●●●
Tolerance for Communication Latency	●●●	●●	●
Need for On-the-fly Component Connections	●●●	●●	●●
CORBA Choice	**ILU**	**TAO**	**Visibroker**

[a]Minor, moderate, and strong factors in CORBA selection are represented by one, two, and three dots, respectively.

Another attribute common to the three SOFIA software products using CORBA is that at least one member of each development team had experience with CORBA or a similar standard. In some cases the experience was limited, and most of the subsequent training was on-the-job, but having some sense of the technology was a consistent starting point.

As seen from Table 1, the DCS, MCS, and FLITECAM each chose a different CORBA product (ILU, TAO, and VisiBroker), with characteristic strengths and weaknesses discussed below. CORBA fit the three systems to different degrees and for different reasons, but each chosen CORBA implementation appeared well suited to its intended use.

CORBA Features Used The principal CORBA features used by these three systems are shown in Table 2. For all three systems the list is quite short, due in part to the systems' early development phase and their developers' CORBA experience. However, this also reflects the fact that while CORBA is very powerful, for any given project only a subset of its features may be useful. In the end, a given data system may not leverage as much of the entire CORBA package as might be expected.

4.1. Data Cycle System (DCS)

The Data Cycle System will support science operations over the entire 20-year operational lifetime of SOFIA, operating more or less continuously throughout that time. (Although it has ground-based and airborne components, we focus

Table 2. CORBA Features used by SOFIA Systems

Function	DCS	MCS	FLITECAM
Name Service/Registration	–	YES	YES
Data (De-)Marshalling	YES	–	YES
Language Portability	YES	–	–
Remote Object Services	YES	–	YES
Connections to Remote Objects	YES	YES	YES

here on the ground-based components.) It will provide an interface for scientists to gain access to SOFIA (e.g., via proposal preparation) and its archived data.

With these requirements in mind, the DCS was designed to collect software on the fly, at user request. For a given data pipeline, algorithms from sites such as Ames, Goddard, and Australia might be combined to reduce archived raw data; while for a proposal, submission components that submit data, automatically review it and provide scheduling metrics could be combined. Even more impressive, new software components can be developed, tested and integrated while existing versions of those components remain fully available to users.

DCS CORBA Motivators Since DCS designers were already familiar with other inter-process and inter-machine communication systems such as Remote Procedure Calls (RPC), moving communications from a procedural abstraction to CORBA's object abstraction was straightforward. Additionally, they anticipated a requirement to support many implementation languages: whereas one instrument team's chosen language might be Research System's IDL, another instrument builder might favor Java or C++.

The DCS will be a widely distributed system, not only spanning local high-speed networks but also taking advantage of possibly distant computer resources. A flexible and robust communications layer was therefore a crucial element of the DCS, and this is a CORBA strength.

DCS CORBA Implementation Process The DCS group performed a trade study with RPC, DCE, CORBA and DCOM, choosing CORBA as the best fit for their requirements. Following a small proof of concept using the Perl scripting language, they began building key products and designing their interfaces with the CORBA product ILU (Inter-Language Unification). (Much of the underlying DCS software to date has been developed in C, an interface for which ILU had implemented early on.)

Benefits of ILU for DCS Promising features of ILU included its free source code (valuable when troubleshooting), multiple underlying protocols, and its multiple language support (a strong plus for the DCS). The flexibility of the product was also impressive, as ILU has easily supported a wide range of functions.

DCS CORBA Current Issues/Regrets The fact that ILU does not currently support the CORBA 2 standard may eventually become a problem for the DCS. Due to its "Erector set" approach (picking tools and features to use as needed),

there is more internal machinery to consider. Finally, it seems likely that using a single CORBA implementation between the MCS and DCS will considerably simplify SOFIA's software maintenance effort, and this will be more carefully evaluated in coming months.

DCS CORBA Status and Future The DCS team has already implemented a sophisticated prototype that addresses certain DCS requirements using multiple computers. Many capabilities must be added to produce the real DCS system, and other CORBA features are likely to be needed for that product. Investigations will be performed to identify and evaluate potentially useful features and to consider the tradeoffs going from ILU to the MCS CORBA product.

4.2. Mission Control Subsystem (MCS)

The job of the MCS can be summarized as routing and displaying data, accepting, routing and responding to commands, performing calculations on data (for example on coordinate systems) and coordinating the observatory and its subsystems. This involves a lot of communication across multiple machines, the state of which may change at any time for reasons both planned and unforeseen (e.g., at 41,000 feet cosmic rays noticeably affect standard workstation memory, with unfortunate consequences for operating system reliability). The MCS is therefore built on the Jini model, with self-registering and self-discovering software components that make connections to each other as needed to perform their allocated functions.

The MCS hardware architecture has evolved over its first two years of development. Whereas multiple processor types (Sun SPARC, Motorola PowerPC) and operating systems (Unix, VxWorks) were originally expected to co-exist, it appears Sun processors with Unix (and its real-time features) may be sufficient to meet SOFIA's needs.

MCS CORBA Motivators Initially, the MCS team was not motivated to use CORBA at all. However, CORBA seemed to address several concerns of the MCS, especially given the likely need to support multiple platforms and operating systems. The ability to connect different processes at run time (and in any order, if the custom-developed code supports it) was particularly appealing. One staff member had overview knowledge of CORBA, while another was very interested in using it, and the team hoped to get significant code reuse by adopting it.

MCS CORBA Implementation Process A "hero programmer" on the MCS team developed a demonstration application using Visibroker in a little over three weeks. The programmer had no previous experience with CORBA, but was very interested in using it. Based on the evident functionality, the project leaders decided to proceed using CORBA tools.

The team purchased the Visigenix VisiBroker product in order to make use of its excellent documentation (the planned CORBA tool had no documentation available during its beta phase), and the first three months of MCS implementation used VisiBroker. Once The ACE ORB (TAO) was formally released, the team replaced the Visibroker software with TAO, encountering only minor issues. Partly reflecting this experience, the team encapsulated MCS CORBA

use as much as possible, so that another product, or custom code, can replace it if necessary. As a result of this encapsulation and the limited CORBA use, the MCS team needs only one or two experienced CORBA developers, whereas three or four would be more typical for a twelve person team.

Benefits of TAO for MCS Like ILU, TAO is freely distributable and its source code is available (Graybeal et al. 2000). It has been extremely stable, with no bugs observed since its first release. It complies with many of the features in CORBA 2.4, the most recent standard. An especially valuable feature is TAO's low communication the—overhead TAO developers took into account real-time data distribution issues during TAO's development.

MCS CORBA Current Issues/Regrets The MCS does not take advantage of CORBA's many features, notably event channels, data management, and other cross-platform features. (The lack of an access-by-value mechanism caused significant latency problems for the data-driven MCS.) Despite the team's focus on rapid data transmission, it is still possible that CORBA and/or TAO introduces too much latency to these communications.

Another potential issue is the criticality of the CORBA Name Server in the MCS system. A failure of this component would force the entire MCS to be restarted, and a suitable backup mechanism has not yet been designed.

MCS CORBA Status and Future The MCS currently uses only a few specific CORBA functions but thoroughly depends on those functions. Future work includes addressing the issues identified above and might either increase CORBA use or eliminate it entirely.

As a specific example of the alternatives, consider the MCS data handling mechanism. Initially, the MCS design rejected CORBA's data marshaling/demarshaling capabilities: the team didn't want to access data across machines. The lack of a pass-by-value feature meant up to six handshakes were required for each data item, and the packing/unpacking process seemed likely to be too slow for the system's requirements. Since the MCS needs to know and manage data structures anyway in order to input and output data, CORBA wasn't expected to add much value.

However, in the current MCS implementation the data structure is unpacked and repacked on every machine in the communication path, and the original format of the data is ignored. On top of that, pass-by-value features are becoming available in CORBA. As a result the current design assumes the overhead burden of CORBA but gains little from its data distribution and management capabilities. This design will be revisited as MCS development goes forward, to take into account lessons learned and improvement in the tools.

4.3. First Light Infrared Test Camera (FLITECAM)

The First Light Instrument Test Experiment Camera (FLITECAM) has a relatively complicated architecture with many components that are selectable during FLITECAM operation. This architecture is designed to maximize flexibility, component reusability and distributability to an unusual degree, making it more like the highly distributed MCS or DCS than a classic monolithic system for a science instrument.

FLITECAM CORBA Motivators FLITECAM has a multi-tier distributed architecture with many small servers. The goals of this design were to support remote instrument control, and to allow redistributing processes to other CPUs as desired. It is an intensely object-oriented architecture, with a heavy Java emphasis, but support for C++ is also necessary. Finally, the developer wanted flexibility in his selection of Java development environments, which meant development and testing might take place on any platform.

FLITECAM CORBA Implementation Process At the outset, the FLITECAM software developer had no CORBA experience. He initially compared CORBA to a solution using RMI/JNI (Remote Method Invocation/Java Native Interface) by developing prototypes using each protocol. He chose CORBA as being the easier to use, especially for certain programming tasks such as exception handling. Immediately thereafter he started building his final product, using a Windows-based development environment.

Benefits of VisiBroker for FLITECAM Visibroker's integration with the chosen development environment, Java IDE (JBuilder 3.5) running on Windows, is extremely good. The developer found that development on Windows and operation on Solaris is feasible using this tool. The Visibroker CORBA solution has proven more powerful and simpler to use than RMI/JNI. Earlier platform independence issues appear to be fully resolved.

FLITECAM CORBA Current Issues/Regrets An early problem with Visibroker's version compatibility (Solaris vs. Windows platforms) caused some worry about future inconsistencies, but those have declined. The developer also feels some need for more in-depth understanding of CORBA's capabilities.

FLITECAM CORBA Status and Future Product development for FLITECAM is well under way and will soon begin core software integration with an external deliverable (containing many C++ modules). In addition, CORBA-related development will focus on pursuing a new event channel model.

5. Conclusions and Recommendations

5.1. Whether to Use CORBA

Does it make sense to use a distributed object services product such as CORBA? Most projects will fall clearly on one side or the other of the CORBA benefit-vs-risk equation. As this paper describes, run-time application coordination in an object-oriented, distributed environment will benefit from CORBA middleware, a conclusion tempered to some degree by whether or not the project has hard real-time requirements. If a data system runs strictly on a single computer, is likely to remain fundamentally stable for several years or already must support well defined interfaces between all of its subsystems, CORBA usefulness will be limited at best.

If the decision is not so clear, the authors believe that a bias in favor of trying CORBA is appropriate. This conclusion is based on four points:

- a reasonably competent software developer can learn and apply CORBA basics in a short period of time (less than a month);
- only essential or useful CORBA elements need be adopted;
- developers can replace CORBA features that prove unsatisfactory;
- it is difficult to evaluate CORBA's value without using it.

Of course, developers should expect to re-evaluate their initial decision once they have some CORBA experience, but changes of heart will not be catastrophic. With the standard's maturity and the release of public domain CORBA products, deciding about CORBA is only slightly more consequential than any other development tool decision.

5.2. Tips When Introducing CORBA

These lessons may be helpful when beginning a CORBA installation.

Seek Out Good Documentation As an example, the TAO CORBA documentation is more readable than the OMG documentation, and the documentation that comes with the VisiBroker product is also good. By now there are also many books on the various CORBA implementations.

Obtain Training As Needed Developers unfamiliar with CORBA may benefit from a short training, say three to five days. Once using the CORBA product, they will quickly go beyond most training classes.

Have Expertise at Hand When something doesn't work, it will be important to have someone to ask. While products such as TAO have responsive mailing lists, the MCS team also benefited from occasional access to CORBA experts.

Encapsulate CORBA Services In a good object oriented design, not every object will need CORBA's services; and those that do should inherit those services where possible. As a result, the developers of most of the data system's objects should not be writing CORBA code directly, and the system should minimize the number of objects using CORBA services. This approach is especially useful if one is unsure whether or not CORBA is the right tool for a data system's development.

References

Graybeal, J., Brock, D., & Papke, B. 2000, Proc. SPIE Vol. 4009-17, "The Use of Open Source Software for SOFIA's Airborne Data System"

Papke, B., Graybeal, J., & Brock, D., 2000, Proc. SPIE Vol. 4014-35, "An extensible and flexible architecture for the SOFIA Mission Controls and Communications System"