CONTROL SYSTEM SOFTWARE ARCHITECTURE R&D PROPOSAL

ROBERT C. SASS

23-AUG-1999

1 INTRODUCTION

This document is a detailed follow-up to the NLC Architecture talk on S/W Architecture R&D presented on 28 July 1999.

With the emergence of the Web and the Internet as the main forces pushing the evolution of computer technology, there are a host of new and rapidly evolving software tools and methodologies that will surely be a central part of the future NLC control system. We have virtually no experience or expertise using any of these tools and methodologies.

A central part of any control system is the user interface that must be able to interact with the rest of the entire system. Here too the possibilities have greatly expanded and we need a whole new look at what is possible and desirable.

The specific R&D items that follow focus on the OPI or GUI and all of the “middleware” that connects it to the rest of the functions and data in the system. Networks, IOC/EPICS software and high-speed pulse-based data acquisition are covered in a separate proposal.

2 R&D OVERVIEW

Figure 1 on the following page gives a general picture of the areas to be covered by this proposal. We divide all of pieces in Figure 1 into 5 main topics. No implementation order implied.

1. Infrastructure. These are the tools to design, build, configure and manage the development and deployment of all of this software.

2. Object/Relational Database. This may be Oracle alone or Oracle plus a pure OO database. All of the definitions for the Control System are stored in these database(s) in one form or another; relational or object.

3. NLC EDD/DM. The display builder and runtime support.

4. CORBA/DCOM. The object bus by which all of the objects interact.

5. Component Implementation. Select a set of functions from our NLC requirements and implement them using all of the database, tools and methodology described above.

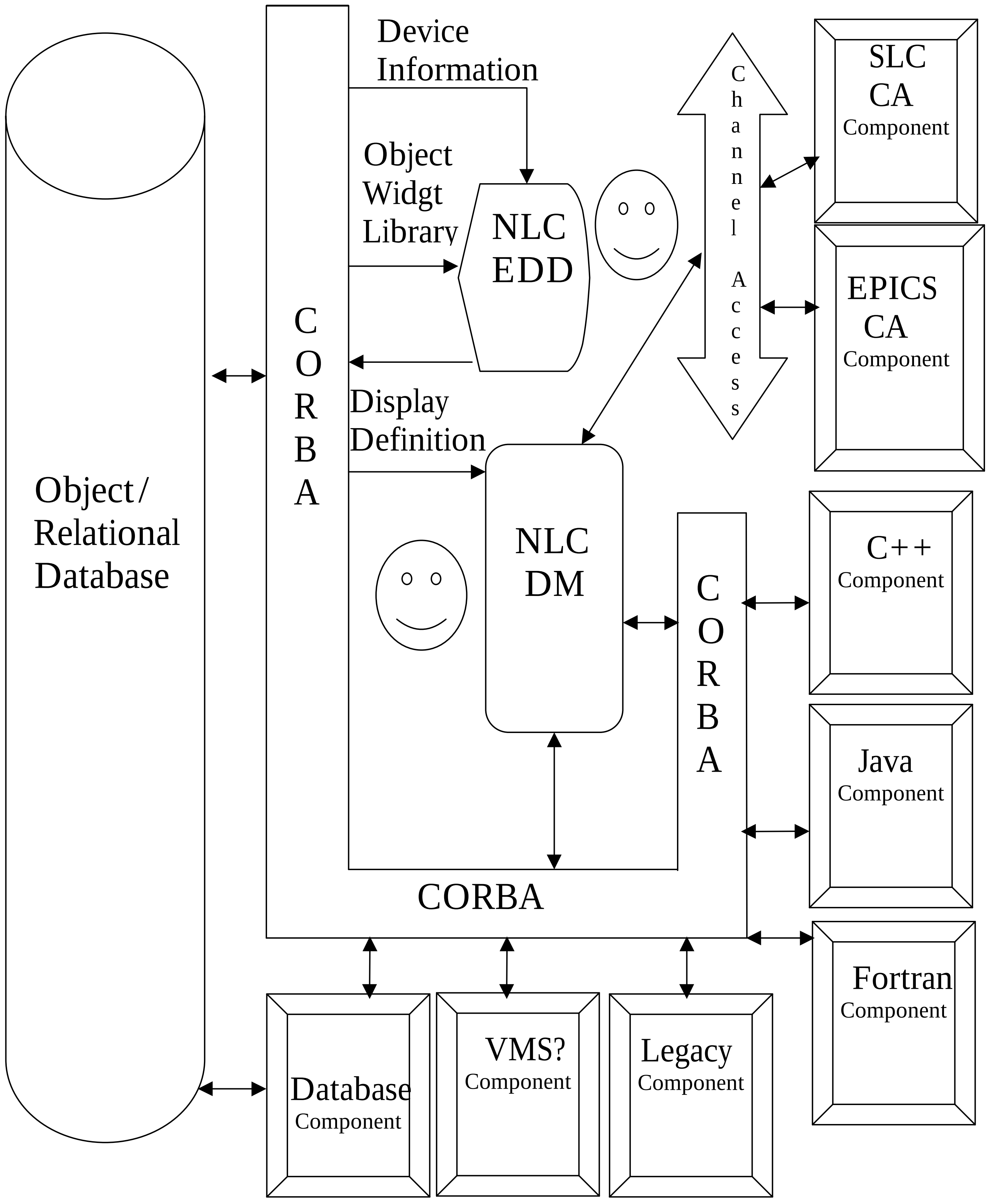

Figure 1 NLC Software Architecture Overview

3 R&D TOPIC DETAILS

3.1 INFRASTRUCTURE

3.1.1 EDUCATION

We need a lot of it since virtually all of this technology is new to us. However, we're fairly bright folks and much can be acquired through reading, self-study, UC extension courses, selected vendor courses and various CBT courses available through Stanford and elsewhere. Some basics should be taught to the group as we've done with C and C++ in the past.

We have virtually no experience in the area of object-oriented requirements, analysis, design and implementation. We should definitely have at lease two classes and perhaps three for the whole group. Since the Unified Modeling Language has emerged as a more-or-less universal graphical standard for expressing OO designs, any course we have should include the use of UML as part of its structure. In conjunction with this, we should also select a common design tool which we should also be used by all of our collaborators.

Mentorship by someone experienced in the art is invaluable. No matter how extensive the class, having someone with experience that can help us with our designs will be invaluable.

The Oracle Database has undergone a major change of direction with release 8i. It now includes both CORBA and a JVM. Release8 itself has several new features that are of interest to us. Along with this are changes to the Designer/Developer tool suite to support the new OO capabilities of release 8.

Education Task List

1. OO Requirements class with UML. We should probably send one or two people to a brief class so we can manage our requirements properly since they are likely to be voluminous. This task would also investigate the utility of using a Requirements Analysis tool like Requisite-Pro.

2. Java Class. Decide if a class is necessary or if we just provide books and some time to read them. The Stanford CBT classes to which we all have access are very good and may be sufficient. Regardless, time must be set aside for self-study.

3. OO analysis class with UML and the selected analysis tool. Emphasize analysis with an eye toward implementing a distributed system using CORBA. This should be a group-wide class held just before we begin real analysis.

4. OO design class with UML and the selected design tool. Again we want to emphasize doing not just OO design but distributed design and implementation. Same caveat as 3.

5. OO implementation class in Java. May be combined with the design class.

6. Oracle 8i and Designer/Developer changes. This task should look at the documentation and class list and see which people and classes are appropriate or necessary. Perhaps just self-study or CBT classes would be sufficient.

7. Hire/contract at least one person with experience in designing and implementing a distributed real-time component system. Yeah, right. This may not be so easy since much of the technology is so new. At a minimum we need someone with real OO design and implementation experience.

3.1.2 PROGRAM DEVELOPMENT TOOLS

We need an analysis and design tool that supports UML. So that we have a common graphical framework and semantics for review and analysis, this tool will be used by all of our collaborators. Because some of our applications are soft real-time, this tool should have added syntax and semantics to support real-time designs.

There are also tools to support code coverage testing, performance profiling and expanded syntax checking beyond that which a compiler normally does.

It might be nice to have a common program editor so that formatting and indenting are common throughout the system. EMACS is certainly one possibility and it's available on just about every platform except our old friend, VMS.

Then there's the question of the actual program development environment to use like Visual J++ and C++ by Microsoft or similar products by Symantec etc.

Program Development Tools Task List

1. Investigate and evaluate various UML analysis and design tools for building distributed OO systems. These may be individual tools or part of a larger suite.

2. Evaluate and select perhaps 2 Java and C++ compile and test tools. Look at Gnu as well as commercial products by Symantec, Inprise (Borland) etc.

3. Survey and make a recommendation for a code management system. All of EPICS uses CVS plus perhaps a Tcl/Tk front end. I know that the EPICS philosophy is "if it ain't free, it's no good" but we should look at Rational **Clearcase** and other products to see if their functionality justifies the prices. Again, distributed S/W maintenance must be considered.

4. Software release control like we have for the SLC only for the distributed non-EPICS software must be put in place. Can we have a common methodology in and out of EPICS? Would be preferable.

5. HyperNews. BaBar has used this very successfully for communication within the collaboration. We should examine their system and perhaps others and then implement a news system for communication between all members of the collaboraton.

3.1.3 SYSTEM MANAGEMENT TOOLS

I get the impression that deployment and management tools for distributed object systems are still in their infancy. We need to find out what's there and write some tools of our own if none fit the bill. Part of the scope of this work is also to define how we are going to manage this development.

System Management Tools Task List

1) Evaluate the current market for distributed system management tools.
2) Write a requirements spec for our management tools needs. What version is running where? How is the system reconfigured to add new nodes etc.?
3) Implement some subset of system management functions for use during the R & D phase.

3.1.4 TEST HARDWARE

Since performance will be one of the critical items on our test list, we need to set up a test environment that in miniature that is reasonably representative of the system configuration we expect to have. The network should be isolated from the rest of our environment with access probably via a gateway as we have done before. We'll be performance and stress testing along the way and we don't want to flood production systems.

Test Hardware Task List

1) One NT Server.
2) One UNIX server.
3) One LINUX Server?
4) One Oracle server instance just for our use but maintained by SCS.
5) Several test workstations. One each UNIX Solaris and NT.
6) Several IOCs.
7) Local, isolated network connecting everything.
8) Large (100 Gbyte?) data storage system for testing history buffers and error logging. Consult with SCS and BaBar on a test configuration.

3.2 OBJECT/RELATIONAL DATABASE

As is clear from Figure 1, one or more databases are central to the entire Architecture. It is the permanent store for just about everything in the Control system. Furthermore, display and dynamic update of database information must be seamlessly integrated on any NLC display.

We need to decide if Oracle alone is enough for our needs or if we should consider a "pure" OO database for some aspects of our persistent data storage needs. We also need to understand clearly how database access fits into the overall scheme of CORBA-based distributed object architecture.

The release of Oracle 8i fits in nicely with the rest of a CORBA-based distributed component architecture and we need to integrate our database functions in a manner consistent with the rest of our implementations. However, it is only the first release of what is sure to be a rapidly evolving set of features. We need to understand where this is going so we don't do an implementation that needs to be redone next year.

Object/Relational Database Task List

1) Get a local instance of 8i just for our use. Since we will be stress and performance testing at various times, we need to be isolated from SCS and the rest of SLAC. Remember what BaBar did to the networks.

2) Thoroughly evaluate 8i and its capabilities for implementing database components using Enterprise Java Beans and CORBA.

3) Test & evaluate table partitioning and indexing improvements for large tables.

4) Evaluate and define how we can use the File System based on Oracle Tables.

5) Evaluate how we can use the extensive effort made by CERN and other labs in integrating Oracle into their Control Systems.

6) Define how we can build complete sophisticated applications around 8i and our display capabilities. Display and interaction with these components must be seamlessly integrated with the rest of the display environment. This may limit the use of the display building tools that come with Oracle since it's likely that they only build database displays.

7) Performance test 8i for a very large database i.e. Archiving and possibly Message Logging. Decide if a look at an OO database is appropriate.

8) Evaluate OO databases if decided above.

9) Design and implement test data to support the rest of the R & D effort.

10) Continue support of the modeling and device database for the NLC as those efforts evolve.

3.3 NLC EDD/DM

As previously mentioned, the display facility is really the core of this R & D effort. It must provide the capability to view and update ANY data in the Control System combined on a single display.

3.3.1 SYNOPSIS OF CURRENT CAPABILITIES

The X-windows runtime Display Manager named DM & its corresponding Display Editor (EDD) and/or its combined Motif counterpart called MEDM form the primary basis of the GUI for the EPICS system. These editors and their runtime cohorts result in displays that consist of a collection of graphical objects that display and/or change the values of EPICS process variables (PVs) via the Channel Access protocol.

The DP tool allows the display of a wildcard list of PVs in tabular form but it is separate from the previous tools and there can be no combination of graphical objects and tabular lists in a single display nor can DM displays reference DP displays and visa versa.

In addition to these basic tools there are Channel Access (CA) interfaces to TCL/TK, the Wingz spreadsheet analysis tool, Mathematica and an indirect interface via TCL/TK to IDL, a scientific visualization tool. There may be yet other interfaces not listed in the standard documentation set.

All of these tools only interface to Channel Access. If you want to combine PV information with data from other sources like commercial databases or calculation results from programs outside of EPICS that don't have a PV, you're plum out of luck.

3.3.2 NLC EDD/DM WISH LIST

One of our tasks will be to make a detailed requirements list of the capabilities desired for this next generation of display generation and management. You can find a list from previous EPICS collaboration meetings. In addition we have more of our own. I'll just cover a few general areas to give an idea of the possibilities so you see that this will be a major effort.

As an example of things one can do with small but sophisticated Java applets, check out Ken Perlin's home page. In particular the zoom capabilities which allows drilldown through graphical displays of Web pages or Neural Networks gives you many ideas as to the kind of sophistication we should come to expect in our operator displays.

3.3.2.1 A Database of Displays

The NLC is likely at least hundreds if not thousands of pages of displays. Furthermore, we need to give Operators and Physicists the capability to create some displays on the fly to meet the needs of the moment. Finding the display you want that has the function you need is sometime a problem even with the current touch panels. Therefore, in addition to links and hierarchies, we should have a searchable database of display information so one can quickly find the page of interest. Thus the main page might be a Yahoo-like list of major functions plus a keyword search capability. There are many possibilities here which need to be explored.

Another option is to have a dynamically constructed visual hierarchy of displays and their functions. One can then drill-down into this hierarchy to find the desired page.

3.3.2.2 Widget Macros

Button Macros are an indispensable tool in current accelerator operations and a similar capability will be needed for the NLC.

3.3.2.3 Master Pages

To maintain a common look-and-feel we should have master pages that can be applied to a defined set of pages.

3.3.3 IMPLEMENTATION FRAMEWORK

In a distributed object system, the display is just the container for a set of active objects. These objects on the display mostly interact with other objects in the distributed system although they can also interact among themselves.

As a famous Physicist who wrote Java Analysis Studio once said if you're going to do a GUI, it doesn't make sense to do it in anything but Java. I'll just quickly reiterate a few of the reasons that you're all familiar with by now:

- Write once run (debug) everywhere. New releases have actually gotten much better in this regard. Tony's experience was that those who wrote in Java had minimal difficulty in running on a different machine. Those who wrote in C++ gave up and just went to Java.

- Can run in a sandbox like in a browser or standalone.

- Mobile code that can be dynamically loaded as needed.

- Much easier graphics programming than X or motif. Again, newer releases like Java Foundation Classes that include the old AWT, 2-D graphics and the Swing components have greatly expanded the graphics capabilities.

Enterprise Java Beans in combination with CORBA seems to provide all of the capabilities we need to develop a rich display system that can continue to evolve with the technology.

NLC EDD/DM Task List

1) Write a full-scale requirements document. For the kinds of possibilities described above you can see that this will be a major effort, as well it should be. Insure that we include all present and planned current DM capabilities. Of course it must run on a Palm Pilot too! Examine the GUI building capabilities of products like Symantec's Visual Café and Inprise Jbuilder for additional ideas.

2) Obtain the code and documentation of the Java DM implementation done by Gary Cooper et al.

3) Define and design the initial implementation. This should include the database interface and client (display) components that interact with server components of at least the following types:

   - Channel Access
   - Relational database.
   - Object Oriented Database; could be Oracle.
   - Application components via CORBA.

4) Define performance tests.

5) Make a series of “production” releases for use by other R & D efforts.

3.4 CORBA / DCOM

But having a display generator and manager is only part of the problem. Now we need objects and EPICS PVs with which it can interact. This gets us into CORBA and distributed objects. These objects can be newly designed applications, legacy applications given an object front-end, Channel Access events and database interfaces.

CORBA includes a large variety of services, some of which overlap Channel Access and others which we may not need.

CORBA / DCOM Task List

1) Implement one of each CORBA freeware and commercial (Tao & Visigenics / Inprise?) and test for interoperability.

2) Evaluate DCOM / CORBA bridges.

3) Evaluate Microsoft DCOM tools. Billy might win this one too and we must be prepared.

4) Evaluate and define which CORBA services we are likely to need.

5) Evaluate high-level application builders like Iona’s Orbix.

6) Define how we’ll use Enterprise Java Beans i.e. server-side components. There are many aspects to this like Transaction Monitors, security, fault tolerance, use of JARs etc.

7) Construct an operating framework in the test system that can be used to run NLCDM and its interactions with the Database, EPICS and other components.

3.5 COMPONENT IMPLEMENTATION

This comes down to building a few selected parts of the whole system in miniature using all of the development tools, NLCDM and CORBA pieces defined in the preceding sections.

Furthermore, we want to develop test applications on the different major platforms that are presently UNIX and NT. If possible or reasonable we could include VMS which would allow us to perhaps make interfaces to reworked existing SLC applications. This means we'd need a VMS CORBA implementation.

We probably don't have to try the matrix of all combinations of languages and platforms but we do need at least to CORBA/Java on NT & UNIX.

In order to put the whole package together we need some representative NLC functions to implement and test. The WBS Tools & Utilities section provides a representative list of possible projects.

Component Implementation Task List

1) Using the selected tools, do a system design for the NLC control system in ultra miniature.
2) Using the selected tools, do a detailed component design for a small set of representative functions.
3) Define system level performance and throughput tests.
4) Implement at least 2 Java components; NT & UNIX.
5) Implement at least 2 C++ components; NT & UNIX.
6) Implement a legacy component such as modeling.
7) Implement several database components.
8) Configure selected applications to use SLC data via PCAS. If we did the job right, this should not be difficult.
9) Combine the whole shebang into a set of displays representing the NLC Control System in ultra miniature.

4 COMMENTARY ON HIGHER LEVEL TOOLS

To a large extent, we are trying to learn the "assembly language" of building a distributed system. This means coding the basic CORBA API calls into the source code using some editor. But for similar sets of components, these calls are repeated in each component and therefore could be easily generated by higher-level software, just like a compiler generates machine code. While we should investigate what tools are available for application generation, it's very early in this game for CORBA and I suspect that the tools have a long way to go, particularly as the underlying API is being rapidly extended.

The Oracle database tools are another example. They are designed to generate displays to interact with data in the database and have no knowledge about information outside of the database. While we may use them for specialized database displays, we will probably not be able to use them for general display where we need to integrate EPICS, database and other data on a single display.

Of course, one of the reasons Microsoft got where they are today is their superb set of Visual tools for third party developers.

5 CONCLUSION

We can expect the whole business of the Object Web to evolve rapidly over the next few years, just during the time period for our R & D. We will have a Golden Opportunity to see what works and what doesn't. By the time we cast our Architecture in concrete, we'll have hopefully made all of the mistakes and can wisely design a framework that we can build on for the next 20 years or so! That's the Theory.