The LINC-NIRVANA Common Software

Florian Briegel[a], Jürgen Berwein[a], Frank Kittmann[a], Valentin Volchkov[a], Lars Mohr[a], Wolfgang Gaessler[a], Thomas Bertram[b], Steffen Rost[b], Yeping Wang[b]

[a] Max Planck Institute f. Astronomy, Königstuhl 17, 69117 Heidelberg, Germany
[b] University of Cologne, Zülpicher Straße 77, 50937 Cologne, Germany

ABSTRACT

The MPIA is leading an international consortium of institutes in building an instrument called LINC-NIRVANA,[1] the **L**BT **IN**terferometric **C**amera and **N**ear-**IR** / **V**isible **A**daptive **IN**terferometer for **A**stronomy.

LINC-NIRVANA is a Fizeau interferometer for the Large Binocular Telescope doing imaging in the near infrared (J, H, K - band). Multi-conjugated adaptive objects is used to increase sky coverage and to get diffraction limited images over a 2 arcminute field of view.

The LN Common Software provides a software infrastructure common to all partners and consists of a documented collection of common patterns in control systems and of services, which implement those patterns. The heart of LCSW is an object model of controlled devices, implemented as ICE network objects. A code generator creates application from templates for these network objects.

Keywords: LINC-NIRVANA, ICE, CORBA, Common Software, Middleware

1. INTRODUCTION

From the beginning of the LINC-NIRVANA project we have been aware of the complexity of the project (see Figure #1). The distributed control system has to manage five independent closed loops (4 adaptive optics and 1 fringe & flexure tracker), 4 deformable mirrors, 2 infrared detectors, 6 CCD cameras, 134 motors, 82 sensors (temperature, pressure and humidity) and 8 temperature controls.

Figure 1. LINC-NIRVANA Bench & Cryostat

{Further author information: (Send correspondence to Florian Briegel)
Florian Briegel: E-mail: briegel@mpia.de, Telephone: +49 (0)6221 528 337
Juergen Berwein: E-mail: berwein@mpia.de, Telephone: +49 (0)6221 528 324

Advanced Software and Control for Astronomy, edited by Hilton Lewis, Alan Bridger,
Proceedings of SPIE Vol. 6274, 62741M, (2006) · 0277-786X/06/$15 · doi: 10.1117/12.671373

Downloaded from SPIE Digital Library on Terms of Use http://spiedl.org/terms

The development of the instrument software is distributed over several institute in Europe. LCSW was conceived early on as a framework not only to provide a software infrastructure suitable for the area of operation, but also as a general guideline for a uniform architecture.

To reduce the effort expended for development and maintenence, it is highly desirable to construct the framework as a layer on top of a proven middleware technology instead of building from scratch. Such an approach should also ensure to save time and money.

The middleware selection process was performed in 2004 and 2005. The choice of middleware had to be made carefully, as it was to form the cornerstone of a software system that falls or raises with the stability and realibility of the used middleware.

Another goal designing LCSW was to limit the number of tools and libraries that have to be used. This is also the main reason for creating another common software for instrumentation.

2. ARCHITECTURE

The LN Common Software - LCSW is located in between the LN application software (Applications) and shared software on top of the operating systems. In particular, LCSW is based on ICE (Internet Communications Engine), which provides the whole infrastructure for the communication between distributed objects. Whenever possible, LCSW features will be provided using off the shelf components and LCSW itself will provide the packaging and the glue between these components

The following diagram shows the main packages in which LCSW has been subdivided. (see Figure #2).

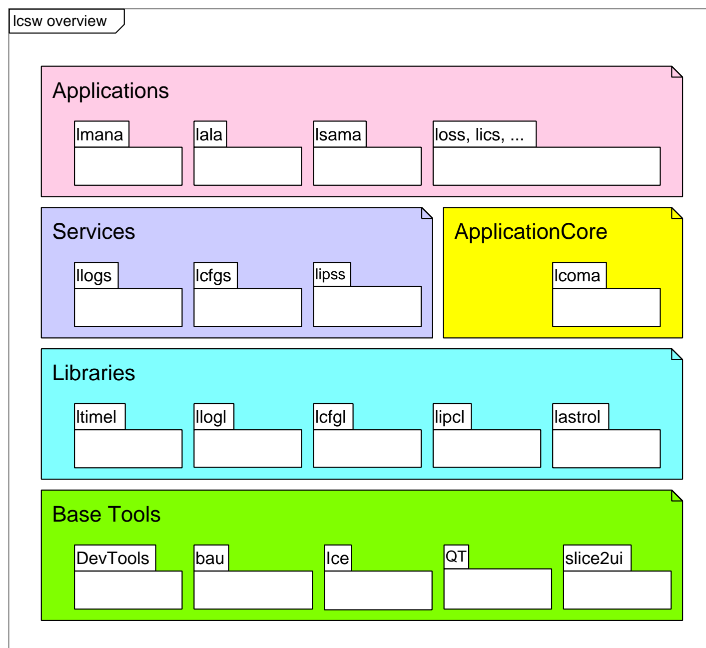

Figure 2. Overview LCSW Packages

1. Base Tools - Mostly 3rdparty Libraries and Programs.
2. Libraries - Libraries for common patterns and wrappers.
3. Services & Application Core

Downloaded from SPIE Digital Library on Aug. Terms of Use http://spiedl.org/terms

(a) Daemons for logging, configuration, remote starting and software distribution.

(b) Core Framework for all LCSW applications.

4. Applications - Manager-, Alarm-, Samplingapplication and other high-level applications from the LINC-NIRVANA project.

2.1. BASE TOOLS

The bottom layer contains base tools that are distributed as part of LCSW or are part of the operating system distribution to provide a uniform development and runtime environment on top of the operating system for all higher layers and applications. These are off-the-shelf components and LCSW itself just provides packaging, installation, building and distribution support. This ensures that all installations of LCSW (development and runtime) will have the same basic set of tools.

2.1.1. Buildsystem - GNU autotools & bau

GNU autotools[2] Autoconf, Automake and Libtool are packages for making your software more portable and to simplify building it–usually on someone else's system. Software portability and effective build systems are crucial aspects of modern software engineering practice. It is unlikely that a software project would be started today with the expectation that the software would run on only one platform. Hardware constraints may change the choice of platform, new astronomical instruments with different kinds of systems may emerge or your vendor might introduce incompatible changes in newer versions of their operating system. In addition, tools that make building software easier and less error prone are valuable.

Autoconf is a tool that makes your packages more portable by performing tests to discover system characteristics before the package is compiled. The source code can then adapt to these differences.

Automake is a tool for generating 'Makefile's (Descriptions of what to build) that conform to a number of standards. Automake substantially simplifies the process of describing the organisation of a package and performs additional functions such as dependency tracking between source files.

Libtool is a command line interface to the compiler and linker that makes it easy to portably generate static and shared libraries, regardless of the platform it is running on.

2.1.2. BAU - bau autotool utility

bau is a small command line utility for creating and managing software projects using autotools. It was created for the LN common software to have a consistent way of building software. Developers of software can create in a very simple way software packages, without deep knowledge of the autotools framework or system dependencies of the various operating systems.

- Built in support for the QT widget library and the ICE middleware.
- Support for automatic documentation (doxygen)
- Packaging as source and binary archives like tar.gz and rpm.
- Helps creating applications from templates as subprojects.
- Support for unittesting and other tests.

2.1.3. Middleware - ICE[3]

ICE, the Internet Communications Engine, is middleware for the practical programmer. A high-performance Internet communications platform, ICE includes a wealth of layered services and plug-ins. ICE consists of the following packages:

Slice The Specification Language for ICE. Slice establishes a contract between clients and servers, and is also used to describe persistent data.

Downloaded from SPIE Digital Library on ... Terms of Use http://spiedl.org/terms

Slice Compilers Slice specifications are compiled into various programming languages. ICE supports C++, Java, Python, PHP, C#, and Visual Basic. ICE clients and servers work together, regardless of the programming language.

ICE The ICE core library. Among many other features, the ICE core library manages all the communication tasks using a highly efficient protocol (including protocol compression and support for both TCP and UDP), provides a flexible thread pool for multi-threaded servers, and additional functionality that supports extreme scalability with potentially millions of ICE objects.

IceUtil A collection of utility functions, such as Unicode handling and thread programming. (C++ only.)

IceSSL A dynamic SSL transport plug-in for the ICE core. It provides authentication, encryption, and message integrity, using the industry-standard SSL protocol.

IcePatch A patching service for software distributions. Keeping software up-to-date is often a tedious task. IcePatch automates updating of individual files as well as complete directory hierarchies. Only files that have changed are downloaded to the client machine, using efficient compression algorithms.

2.1.4. QT[4]

QT is a comprehensive C++ application development framework, which includes a class-library and tools for cross-platform development and internationalisation. The intuitive QT API and tools are consistent across all supported platforms, enabling platform-independent application development and deployment.

The QT Class Library Is a growing library of over 400 C++ classes, which encapsulates all infrastructure needed for end-to-end application development. The elegant QT API includes a mature object model, a rich set of collection classes, and functionality for GUI programming, layout, database programming, networking, XML, internationalisation, OpenGL integration and much more.

QT Designer Is a powerful GUI layout and forms builder, enabling rapid development of high-performance user interfaces with native look and feel across all supported platforms.

2.1.5. QWT,[5] QWT-plot3d,[6] PyQT[7]

QWT is a graphics extension to the QT GUI application framework from Trolltech AS of Norway. It provides a 2D plotting widget and more.

QWTPlot3D is a graphics extension to the QT GUI application framework. It provides a 3D plotting widget for scientific data and mathematical expressions. It compares to the existing QWT Project.

PyQWT = FAST and EASY data plotting for Numerical Python and PyQT. PyQWT is a Python wrapper for the QWT C++ class library which extends the QT framework with widgets to display and control data for scientific and engineering applications.

2.1.6. slice2ui

This tool can generate QT User Interfaces from a slice interface definition file. The user interfaces are very elementary structured. Every interface method is represented as a line. The first widget element is the return value, the second is a button for calling the method and the rest are the method parameters. At present time only basic type are supported (string, integer, float and boolean), call by value and by reference is supported.

In the future there will also be support for the data types sequence and vector.

Downloaded from SPIE Digital Library on ... Terms of Use http://spiedl.org/terms

```
interface ConfigService
{
  double getDouble(string qalName);
  long getLong(string qalName);
  string getString(string qalName);
  void setDouble(string qalName, double val);
  void setLong(string qalName, long val);
  void setString(string qalName, string val);
  void removeKey( string qalName);
};
```

Figure 3. slice2ui - Generating a User Interface from a slice interface definition.

2.2. Libraries

The second layer ensures standard interface patterns for time, logging, configuration, inter process communication and astronomical functionality.

2.2.1. ltimel

Library for everything that has to do with time. Standard time format and conversion functionality for all LN components.

2.2.2. llogl

Logging of data, actions and events. Filtering of log messages by 9 levels of type - Trace, Debug, Info, Notice, Warning, Error, Critical, Alert, Fatal. The filtering can be set at compile- and runtime. For each logging message a unique identifier can be set by the user. For every log message a time stamp in milliseconds, source code information (line & file) and runtime context information (host, process id, & user) is automatically generated.

```
2006-05-12 08:31:20.452923 DEBUG    myThread lisa:23929 briegel ThreadTest.cc:46 void myThread::run() [#2 Counter #19]
2006-05-12 08:31:20.452934 INFO     myThread lisa:23929 briegel ThreadTest.cc:47 void myThread::run() [#2 Counter #19]
2006-05-12 08:31:20.452945 NOTICE   myThread lisa:23929 briegel ThreadTest.cc:48 void myThread::run() [#2 Counter #19]
2006-05-12 08:31:20.452956 WARNING  myThread lisa:23929 briegel ThreadTest.cc:49 void myThread::run() [#2 Counter #19]
2006-05-12 08:31:20.452967 ERROR    myThread lisa:23929 briegel ThreadTest.cc:50 void myThread::run() [#2 Counter #19]
2006-05-12 08:31:20.452978 CRITICAL myThread lisa:23929 briegel ThreadTest.cc:51 void myThread::run() [#2 Counter #19]
2006-05-12 08:31:20.452989 ALERT    myThread lisa:23929 briegel ThreadTest.cc:52 void myThread::run() [#2 Counter #19]
2006-05-12 08:31:20.453000 FATAL    myThread lisa:23929 briegel ThreadTest.cc:53 void myThread::run() [#2 Counter #19]
```

2.2.3. lcfgl

Common API for accesing the LN configuration file format from which LN components retrieve their initial configuration. The configuration data is structured hierarchically as a n-ary tree. Right now the choice was made to use a fits like configuration file format.

```
DCS.VERSION = 0.9 # current version
DCS.MOT1.DESCR = "DCS Derotator"
DCS.MOT1.SPEED = 124.7
DCS.MOT1.MINPOS = 200.0"
DCS.MOT1.MAXPOS = 15000.0"
```

2.2.4. lipcl

The LN Inter Process Communication Library implements patterns[8] and wrappers for synchronous and asynchronous communication between local threads and distributed processes.

- ExceptionStack - Sequence of error objects, which is looped to higher levels.

error(n-1) –> error(n-2) –> error(1) –> error(0)

Downloaded from SPIE Digital Library on Aug 2010 Terms of Use http://spiedl.org/terms

- Wrapper for Singleton, Mutexes and Monitors
- ThreadPool - Class for Life Cycle Handling of threads.
- Reactor[9] - Provides timer events and single-shot events.
- ObjectFinder - Class for finding ICE services by multicast.
- Client/Server - Wrapper classes for easy creation of ICE services.
- Crashhandler - Class for handling segmentation faults. Dumping a stack trace via the GNU debugger and tries to call a user supplied emergency function (closing files, etc).

2.2.5. lastrol

Unified interface to various standard astronomical libraries or programs (e.g. wcstools, sextractor, etc).

2.3. Services

The third layer implements services that are not strictly necessary for the development of prototypes and test applications, or that are meant to allow optimisation of the performances of the system.

2.3.1. llogs

Interfaces and implementation for the logging database for retrieving data, actions and events from LN Components. Filtering, caching and delayed deliver to the central logging service. Tools for displaying and handling. Support for an event based callback interface for consumer applications, with various filtering capacity. Will be used by the sampling system lsama, the alarm system lala, the manager application lmana and others.

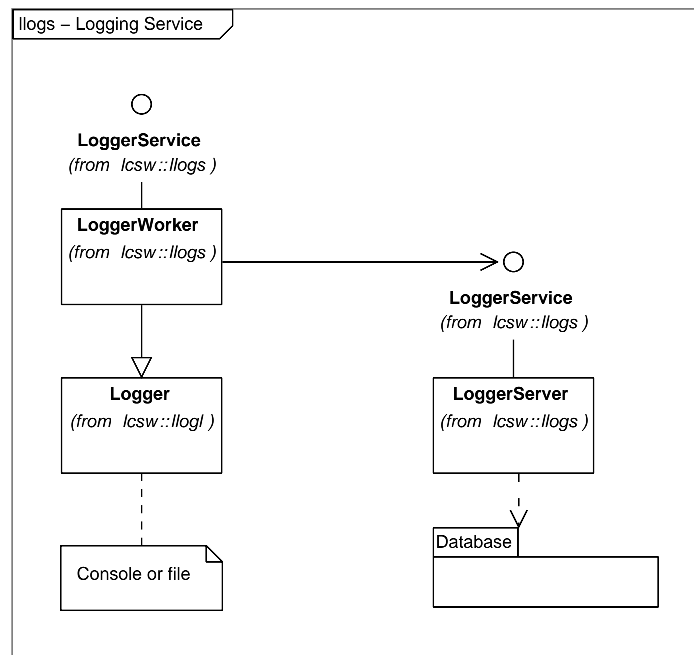

Figure 4. llogs - Logging System

Downloaded from SPIE Digital Library on Aug. Terms of Use http://spiedl.org/terms

LoggerWorker Logging Que owned by a thread which processes all the messages (filtering, delaying, caching and forwarding) of the single- or multithreaded application. The cached log messages can be accessed with the LoggerService interface through polling by time interval or an event based mechanism.

2.3.2. lcfgs

Interfaces and basic implementation for the Configuration Database from which LN Components can retrieve their initial configuration from a central point, additionally to the features from lcfgl (see 2.2.3 on page 5).

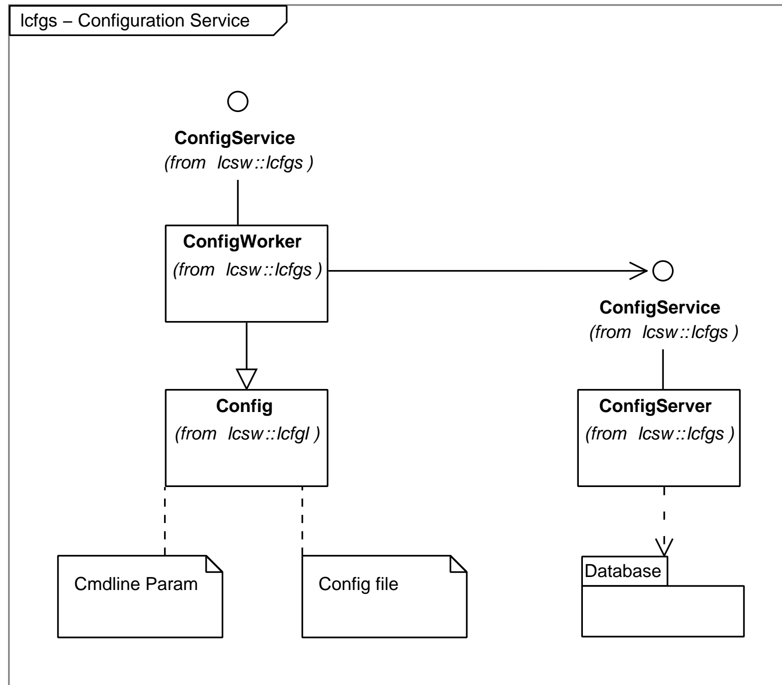

Figure 5. lcfgs - Configuration Service

ConfigServer Multithreaded implementation of a central configuration service accessible through a slice interface.

ConfigWorker Threaded implementation of the applications local configuration tree with a slice interface for remote access to the configuration nodes. Another feature of the remote interface is an event based system for other applications that can register for changes of the values on trees and nodes.

2.3.3. lipss

Service daemon running one each computer once and it has two duties and responsibilities:

1. Based on the ICE patch service it keeps the installed LINC-NIRVANA software and configuration data on the distributed systems in a peer-to-peer manner in sync.
2. Rather than starting applications by ssh or rsh, the lipss daemon provides a remote start interface.

Downloaded from SPIE Digital Library on ... to ... Terms of Use http://spiedl.org/terms

2.3.4. lcoma

This class provides an application framework that allows an easy implementation of a standard LCSW application (skeleton application) with all essential features. Inheritance and other mechanisms are used to enhance and extend the basic application according to specific needs.

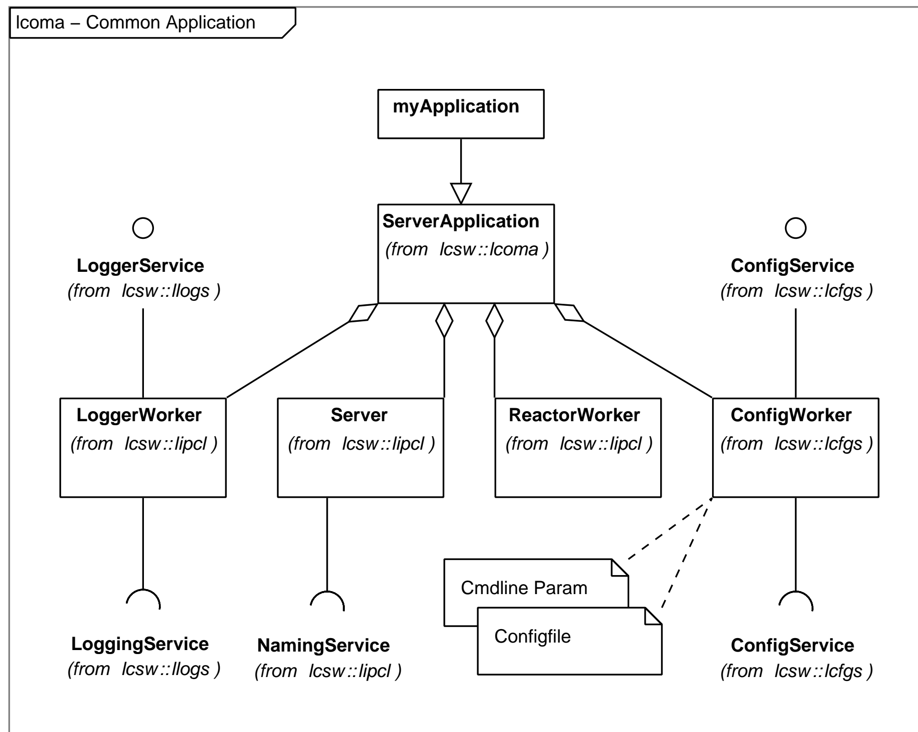

Figure 6. lcoma - Common Server Application Skeleton

Application Skeleton The application framework enforces common solutions and adherence to predifined standards. It will be implemented as a set of classes that provide:

1. Skeleton for standard application.
2. Standard logging mechanism LoggerWorker (see 2.3.1).
3. Standard configuration and runtime properties (see 2.3.2).
4. Implementation of a standard state machine for Life Cycle Handling (see Figure #7).
5. Standard Service Interface for Life Cycle Managment (see 2.4.1).

State Machine

1. Offline - The initial state of a lcoma application
2. Standby - The lcoma application is initialized and ready for action.
3. Online - In this state it is possible to send a command and more

Downloaded from SPIE Digital Library on 1 Aug 2011 to Terms of Use http://spiedl.org/terms

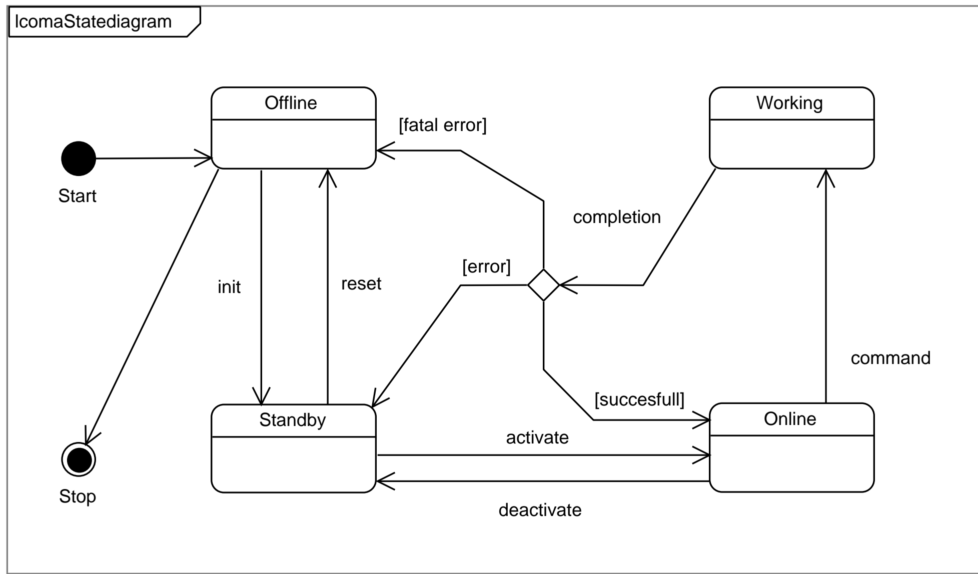

Figure 7. lcoma - Statemachine of all lcoma applications.

4. Working - It is doing hopefully the right thing.
   (a) On occurence of a fatal error the lcoma application should go at once offline (e.g. Hardware failure).
   (b) If a less severe error emerges it should fall back to standby (e.g. Motor hits the endswitch).

2.4. Applications

The fourth and last layer provides high level APIs and tools.

2.4.1. lmana

The LN Managing service application is divided in to parts, a managing daemon and a user interface for administration of the component applications derived from lcoma.

lmana service

- Livecycle managment (see Figure #7) of lcoma applications.
  Deploying an application consisting of multiple server programs on a large network of hosts can quickly degenerate into a nightmare of tedium fraught with accidental discrepancies between systems. lmana Lifecycle Managment relieves much of this pain by allowing you to codify your application's deployment using a repeatable, reliable process.
- Naming service for lcoma applications - Enables clients to discover their servers. By acting as an intermediary, lmana naming service decouples clients from their servers.

lmana user interface This user interface is the main managing panel (see Figure #8) for all lcoma applications running on various computers as processes.

1. Lifecycle Managment of lcoma applications - Start, Stop and change State.

Downloaded from SPIE Digital Library on Aug 2017 Terms of Use http://spiedl.org/terms

2. Overview of lcoma applications running on various computers. Every lcoma service has a unique identifier name, the name is hierachically (similar to DNS). For example in Figure 8 the filter wheel service LN.LIRCS.FILTER was selected.

3. Life Cycle properties for the lipss daemon (see 2.3.3) and status information of the lcoma application.

4. Browser for the selected lcoma application Service owned internal configuration and runtime parameter tree.

5. Realtime logging and history browser of the selected lcoma application Service.

6. Engineering user interfaces of the selected lcoma application Service.

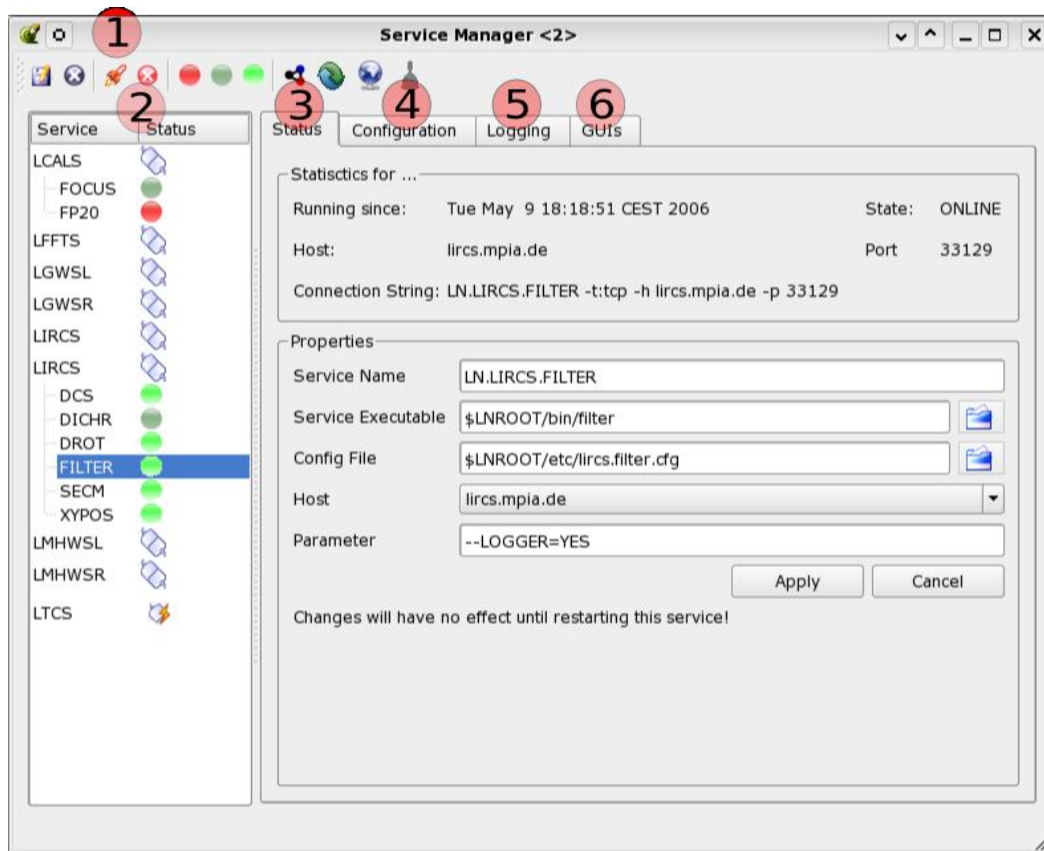

Figure 8. lmana User Interface Lifecycle Managment.

2.4.2. lala

Tools for configuration of hierarchical alarm conditions, API for requesting notification of alarms at the application level, tools for displaying and handling the list of active alarms.

2.4.3. lsams

The sampling system lsams is a high level tool for two- and three-dimensional graphical presentation of data (virtual oscilloscope) from the logging system llogs.

It can be configured to sample one or more parameters at given frequencies directly from the lcoma applications Configuration Service Interface(see 2.3.2) or through the Logging system llogs(see 2.3.1).

Downloaded from SPIE Digital Library on Aug ... Terms of Use http://spiedl.org/terms

3. CONCLUSION

Over the last years, we have seen, evaluated and used various common software frameworks for instrumentation. In general, they work better or worse the same way. Some of them were very basic and others tend to be real monsters in complexity.

Our goal was to create a small but powerful common instrumentation software. To achieve this we decided to use a industry grade object-oriented middleware. The major player in this market are DCOM, CORBA[10] and SOAP.

DCOM was a Microsoft-only solution that could not be used in heterogeneous networks containing machines running on a variety of operating systems. DCOM was superseded by the Microsoft .NET platform.

CORBA is available from a variety of vendors, but it was rarely possible to find a single vendor that could provide an implementation for various programing languages and operating systems. Despite much standardization effort, lack of interoperability between different CORBA implementations caused problems. CORBA suffered extremely from excessive complexity. Becoming proficient and designing for and programming was a sophisticated task that took many months (or even years) to master. CORBA has been stagnating in recent history and a number of vendors have left the market, leaving the customer without support.

Simultaneously with the decline of DCOM and CORBA, a lot of interest arose in the distributed computing community around SOAP and web services. The idea of using WWW infrastructure (XML) to develop a middleware platform was intriguing - at least in theory. Despite much publicity and many published papers, this kind of middleware is not suitable for instrument control. SOAP and web services imposes very serious performance penalties on applications, both in terms of network bandwidth and CPU overhead, to the extend that the technology is unsuitable for many performance-critical systems.

In the end we decided to use the middleware ICE for several reasons:

- Several programming languages are supported: C++, Java, Python, PHP, C#, Visual Basic.
- All major operating systems are supported: Linux, MacOS, Solaris, HP UX, AIX, Windows.
- Excellent performance for both latency and data transfer.
- Small memory footprint $<$ 3 MB shared libraries* .
- Easy to learn.

While LCSW was developed primarily for the LINC-NIRVANA project, it will have its first real world test with another astronomical instrument. LAIWO[12] the Large Area Imager for the Wise Observatory which will be delivered in November 2006.

ACKNOWLEDGMENTS

We are grateful to the entire LINC-NIRVANA team at the Max-Planck Institute in Heidelberg (Germany), MPI for Radioastronomy in Bonn (Germany), University of Cologne (Germany), Astronomical Observatory of Rome in Monte Porzio Catone (Italy) and Arcetri Astrophysical Observatory in Firenze (Italy) for their professional astronomical and engineering support.

*The CORBA implementation TAO[11] requires 120 MB memory, only to run a simple CORBA based application.

Downloaded from SPIE Digital Library on ... Terms of Use http://spiedl.org/terms

REFERENCES

1. T. Herbst, R. Ragazzoni, and Team, "Linc-nirvana - lbt interferometric camera - near-ir visible adaptive interferometer for astronomy." http://www.mpia.de/LINC/index.html.
2. G. V. Vaughan, B. Elliston, T. Tromey, and I. L. Taylor, *GNU Autoconf, Automake and Libtool*, Sams, 2000 (first edition). http://sourceware.org/autobook/.
3. M. Henning and M. Spruiell, "Ice - internet communications engine." http://www.zeroc.com.
4. Various, "Qt sets the standard for high-performance, cross-platform application development." http://www.trolltech.com/.
5. U. Rathmann, "2d plotting widget and more." http://sourceforge.net/projects/qwt.
6. M. Mutschler, "3d plotting widget for scientific data and mathematical expressions." http://sourceforge.net/projects/qwtplot3d.
7. G. Vermeulen, "Python wrapper for the qwt c++." http://sourceforge.net/projects/pyqwt.
8. E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns*, Addison Wesley, 2003 (26. edition).
9. D. C. Schmidt and I. Pyarali, "The design and use of the ace reactor - an object-oriented framework for event demultiplexing."
10. S. V. Michi Henning, *Advanced CORBA Programming with C++*, Addison Wesley, 2004 (9. edition).
11. D. Schmidt, A. Gokhale, T. Harrison, D. Levine, and C. Cleeland, "Tao: A high-performance endsystem architecture for real-time corba," 1997.
12. D. C. Afonso and D. K.-H. Marien, *LAIWO - The Large Area Imager for the Wise Observatory*. http://www.mpia.de/LAIWO/index.html.

Downloaded from SPIE Digital Library on ... Terms of Use http://spiedl.org/terms