The Open Inspire Architecture for Control, Data Reduction and Analysis

Stefan A. Flemming[1+2], Dr. Jonathan A. James[2], Dr. Rainer Schneider[1], Dr. Robert C. Wimpory[1], Dr. Michael Hofmann[3], Martin Schobert[1], Christian Randau[3], Fabian Holstein[1], Thomas Weiss[1]

[1]*Helmholtz-Institute Berlin for Materials and Energy, Glienicker Straße 100, D-14109 Berlin, Germany.* [2] *Open University, Milton Keynes, UK.*
[3]*Forschungsneutronenquelle Heinz Maier-Leibnitz (FRM-II), TU München, Lichtenbergstr.1, D-85747 Garching, Germany.*

October 30, 2008

Abstract

The increasing variety of new experiments being performed on beam line instruments, coupled with the necessity for frequent changes of instrument components gives rise to the need for a flexible control software that can be easily adopted to new experiment environments. Open Inspire solves this demand by an internal IoC[1]-based [1] application container that is responsible for wiring loosely coupled modules to an experiment using an xml-wiring file. Modules for hardware access, scripting in multiple languages, simulation or monitoring can be arranged in arbitrary ways using a graphical wiring editor on a remote client. This approach enables the advantages of using tried and tested instrument control systems to be combined with those inherent in modern software designs. Helpful tools for the complete range from the development to the execution by the user are provided to circumvent as many time-consuming and complex tasks as possible.

1. Introduction

Open Inspire is a scientific control, data reduction and analysis system organized in an n-tier network architecture.
It consists of an Application Server [2] for data reduction and control, several hardware servers to access devices or real-time code and one or more user interface clients for monitoring, configuration and interaction.

The focus is on creating a reliable system with a reusable and scalable component design that provides a bridge between legacy and modern architectures used in instrument control.

The design offers a maximum code lifetime since all modules are loosely coupled [3] and can be refactored or immediately used in other applications without binding it to any part of OI.

Easy and distributed development in teams is achieved by providing a variably grained black box component design. This enables a steep learning curve since knowledge behind the borders of the specific work units is not required. Information hiding reduces the need for deriving from proprietary classes or following complex coding conventions.
Central versioning, a sophisticated build system that provides debugging, refactoring and profiling capabilities and a community based website facilitates development.

2. Technologies and Design

Legacy control systems are often designed in a monolithical way, which makes it

[1] Inversion of Control

difficult for new developers to work with them. Extending or refactoring the sources often takes more time than rewriting the part, which causes many different systems that solve exactly the same problem. On the other hand, new enterprise design paradigms are often overdesigned and try to save the world while covering all imaginable cases.

Open Inspire relies on a clear and lightweight concept and uses information hiding techniques to keep the focus on the relevant development interest without loosing scalability. Using the most common standards helps developers start in a well-known way so that they are investing time in learning a standard rather than a home-brewed solution.

2.1. Network Model

Integrating a new system in a legacy environment is a challenge because maintainability, security and firewall issues, as well as throughput and real-time behaviour have to be considered.
A complete Open Inspire system consists of three tiers: a central Application Server, several device servers and one or more user interface clients. The protocols between the application server and services are not restricted to allow mixing different architectures.

2.2. Data and Control Services

A wrapper server for data or instrument hardware access provides a network service that makes some kind of functionality available. It can be hardware with an integrated network interface, a software based server that wraps the access to hardware, real-time code, legacy code in other languages, a database or an application container on a different host.
There is no rule as to the language in which this server has to be implemented, the protocol that has to be used or what the interface should look like.
This regulation allows the connection of legacy components with known interfaces to Open Inspire as well as services with future protocols.
Furthermore it is possible to mix or replace different technologies as is done in the prototype where CORBA [4] based CARESS [5] device servers are mixed with SOAP [6] based web services and RMI [7] based monitoring.

3. The Inspire Application Server

The Application Server is the central point of the n-tier Inspire architecture and houses its own Application Container, a handler for Open Inspire Modules (OIMs).
Using a central application container for module hosting and middleware [8] solves several issues concerning performance, maintainability, data integrity or security.

3.1. Container Design

The container behaves in a similar way to Java EE [9] Application Containers, except that it is optimized for experiment control tasks.
Versions of Open Inspire prior to Open Inspire 2008 used the *Inversion of Control* Container of the Spring Framework [10] for dependency injection. The new version now provides its own similar IoC-Container, which is more lightweight since it does not need to implement almost all of the web-based technologies and instead provides services necessary for the instrument environment.
Good frameworks do not only have to provide much functionality, they need sophisticated methods of information hiding, restricting source code access and taking care of user roles. Providing a script language with full access to all components for configuration and state handling is not a good idea. Coupling modules with in-code references is another bad idea. Both efforts reduce maintainability, update capabilities and security due to the lack of code separation.
Open Inspire uses the inversion-of-control pattern for wiring loosely coupled modules via dependency-injection inside the container.

Optional global services such as logging, error handling or messaging can be registered against a Java 6 service provider interface in META-INF/services. This enables for either publishing new services or using services published by the container or other modules without hard binding.

3.2. Module Design

An Inspire Module is a standard Java Bean, packaged in a jar-File [11]. There is no need for mandatory library dependencies, inheritance or other types of hard binding to the container.
Inspire identifies a particular module's behavior via additional entries in the manifest of the package and annotations [12] in the Java Beans [13] source code. This makes the module directly available for use in every other application outside the container by simply ignoring this meta-information.
The manifest contains entries for the module name, description, icons, the service provider class and a list of dependencies to other OI-Modules or libraries.
To enable data transfer between modules and ways for external configuration, every module can provide ports and properties in a service class.
A property is a setter-method, marked by the *OIProperty*-Annotation and is used to publish a configuration parameter that can be configured by the container.
A port is also a setter-method, this time marked with the *OIPort*-annotation and used by the container to inject a reference to a different module in the same container.
An optional method *initComponent* can be implemented and will be called by the container if it exists. This method can be used to start threads or do calculations after the properties and ports are available.
In contrast to the *initComponent*-Method a *shutdownComponent*-Method can be called to stop threads, close connections or free references.

3.3. Module Wiring

The Open Inspire Container is responsible for the dynamic wiring and configuration of modules within the context.
An XML-document similar to a spring-context is used to configure the experiment environment with all necessary modules such as proxies for hardware servers, state machines, database connectivity or scripting. The context itself explicitly does not contain any script code to be kept maintainable by external configuration tools. The file validates against the schema http://openinspire.org/schema/wiring/open-inspire-wiring-1.0.xsd and contains all modules for the experiment including their configuration properties and ports (references to other modules).

3.4. Container Lifecycle

When starting a wiring file by the container, a couple of steps will be performed: First of all, the xml-context file is parsed and validated. In case of a valid wiring file, a list with all required modules and dependencies is created. When all dependencies are resolved and available, the module class loader dynamically loads all required classes for the context at runtime. If modules are missing but available in the central Inspire Module Repository, they can be downloaded by the container, otherwise they have to be copied in the modules folder. When all dependencies are resolved, the container creates instances of all service classes and injects the properties and port references for the created objects. Finally a topological sort algorithm is used to sort all modules that implement the *startComponent*-Method and calls these methods in the correct dependency order. This assures that all dependent modules are configured and available before starting a module that uses them.

Shutting down the container is similar to the start-up, except for the fact that the *shutdownComponent*-method is called in reverse order. Here all references have to

be removed, connections closed and threads stopped to allow the garbage collector freeing the memory.

4. User Interface

The last missing part in the environment is a graphical user interface since the server runs as a daemon and does not provide more than a System Tray [14] icon for container management and a module for visual debugging, usable on graphical systems.

The OIDG [2]does not make regulations on how a client should look like. It is even possible to choose the network protocol, because it is registered against a service interface. This enables the server to be used in combination with clients in different languages or architectures as well as web based thin clients.

4.1. Open Inspire Sunrise

Open Inspire provides an implementation of a graphical user interface called Open Inspire Sunrise. It is built with Java, on top of the Netbeans Rich Client Platform [15], which provides a good code base for every scientific application. Using a rich client platform saves much time in contrast to developments from scratch and assures the usage of suitable standards. Netbeans is directly supported by Sun[3], builds on standards, has been perfected in 10 development years and inspired many other RCP and IDE [4]developments. Sunrise uses Swing in combination with VisAD [16] for all graphical elements and renderings. Swing is the standard widget toolkit for Java, providing the largest set of visual elements uses native OS routines for renderings newly provides real system look and feel. The VisAD library provides 2D- and 3D- capabilities, used for all scientific visualization and delivers data structures and calculation tools.

[2] Open Inspire Design Guidelines
[3] Sun Microsystems Inc.
[4] Integrated Development Environment

4.2. Network

Using a client in the same subnet as the server is usually no problem but monitoring an instrument from other networks can be complicated.
The option to use an instrument from the office, at home or during travels requires additional security precautions.
Open Inspire provides user authentication and roles to restrict the container access and manipulation of loaded modules. Often this is insufficient and the instrument network is secured by a firewall, which exclusively allows access through proxies or tunnels. OI is optimized to work in firewalled environments and provides mechanisms to make this as easy as possible. The complete communication is handled by one selectable port that can be tunneled instead of dynamic ports or ranges. If SSH-tunneling [17] is demanded, the Sunrise Login Dialog provides direct support.

Figure 1 The Sunrise Login Dialog

A different and common problem however is that the firewall settings of the client prohibit incoming connections and make asynchronous callbacks to the client impossible.
Inspire solves this problem by using a thread that initiates an outgoing call to the server and blocks until new data is available for the client. If a new message is available, the thread wakes up and propagates the message to all client

listeners. The latest OI version uses the *ClientProxy* to implement this feature with the Cajo-Framework [18].

4.3. Remote Wiring

Inspire implements a remote file system that enables server side wiring files to be accessed remotely. All wiring files are organized in a tree structure and can be created, copied, moved, renamed or deleted as if they were on the local computer. A selected context file can be started and stopped remotely using its context menu or a toolbar entry.

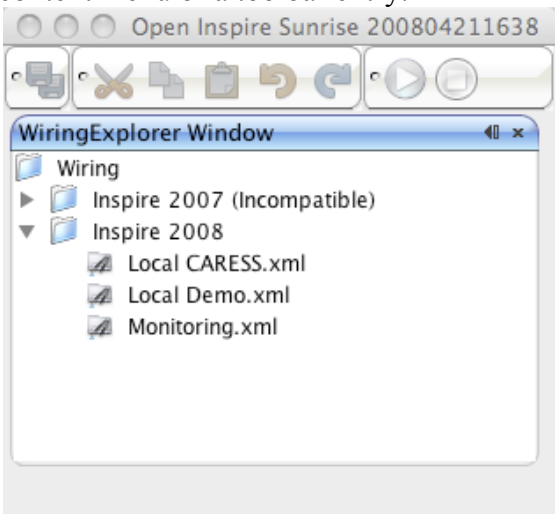

Figure 2 The Remote Wiring Explorer

Creating and editing wiring files is also supported. When a valid wiring file is opened it can be edited remotely either in a textual or graphical way. Graphical editing is the recommended way since it is more fail-safe and faster. Installed modules can be dropped from a palette to the wiring window, configured in the properties window and connected with other modules.

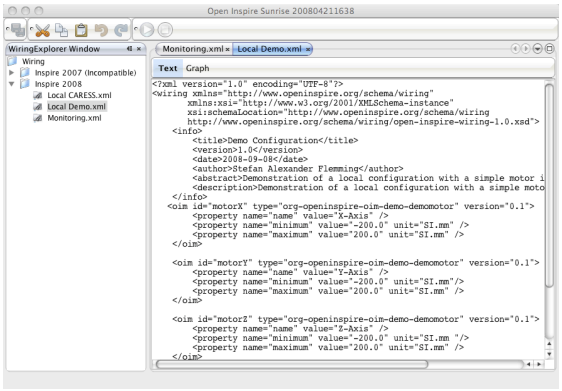

Figure 3 The Sunrise remote wiring text editor

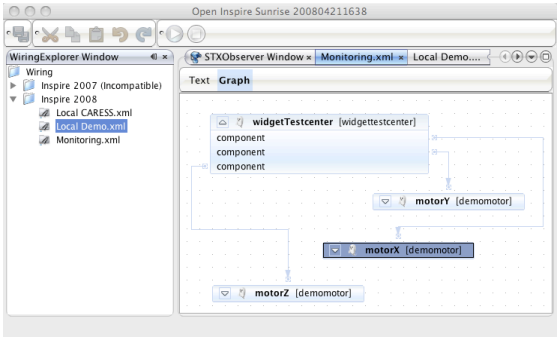

Figure 4 The graphical Sunrise Wiring Editor

4.4. Remote Service Lookup

Modules in the container are able to register services against a central service lookup, which is used by authenticated clients.

If a service becomes available in the Lookup and a client extension is installed that listens to this service, new features such as tool bars, menu entries, windows or data visualization are dynamically added to the user interface. An additional switch can be set by the service provider to enable or disable a service without deregistering it.

5. Modules in Action

This section demonstrates a few important Open Inspire modules, used in a real instrument environment.

The Helmholtz Centre Berlin for Materials and Energy operates three Stress and Strain Scanners with similar setups. All instruments consist of a sample table with three translation axes xT, yT and zT and a rotation table OMGS. A single detector

SDET is mounted in front of the beam stop and a 2D-detector ADET can be driven around a TTHS circle around the sample table. Internal high precision counters are used to count incoming neutrons and time.
Latterly a camera has been mounted for Open Inspire, used by the instrument alignment module to determine a precise instrument calibration and the sample orientation.

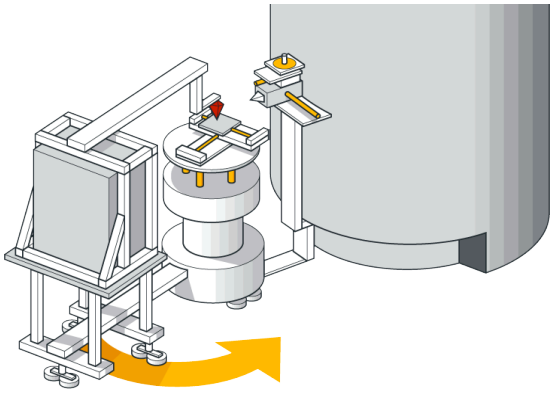

Figure 5 A figure of the reference instrument

5.1. Hardware Access

Accessing the instrument hardware is done by wrapper modules, which encapsulate the remote access to a corresponding device. These modules provide a local proxy interface to other modules in the container that hides all implementation details.
Unfortunately there is, except for a few proposals such as the Grand Unified Model [19], no international standard for interfaces that can be used to access devices of the same type in different control systems.
Inspire provides a unique set of interfaces in the module *DomainScience* for common scientific devices such as motors, detectors, counters, cameras, state machines or simple values. Putting these interfaces in a standalone module enables it to be substituted when a standard finally exists. This also allows the implementation more than one standard interface for a device and decouples the interface description and framework.

The instrument in this example mixes CORBA based CARESS device access with SOAP based web services.

The modules *CaressMotor*, *CaressCounter* and Caress Detector wrap the CORBA communication to the CARESS server and provide standard interfaces to the container.

Another module is the camera module, which wraps the SOAP based access to the Inspire Camera Service and provides access to the picture, the edge positions of the sample and methods used to set parameters like threshold, contrast and detection methods.

5.2. Monitoring

The observation of instrument and experimental parameters is essential for monitoring the health and status of the setup, either directly at the instrument or remotely.
There are currently two ways to realize this. The first way is explained in the Debugging section and uses a module that accepts any number of connections of arbitrary type. Whenever a widget is available that fits the interface of a connected module an element will be added to the user interface, which can be used for monitoring and control.

A more common way is to write a module with a number of fixed ports for modules that should be monitored. This enables the creation of a matching user interface for Sunrise with a more descriptive view.

One example is the module *STXMonitor* that combines information about axes, detector views, camera pictures or values of Euler Cradles with pleasant graphics.

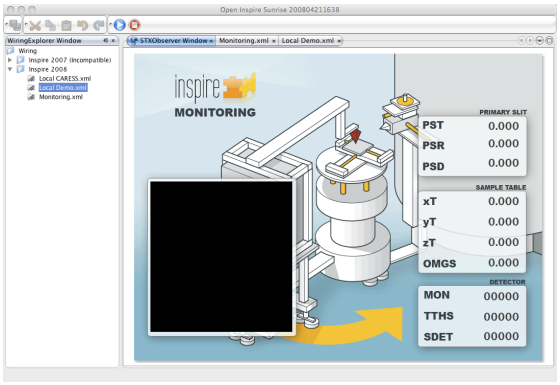

Figure 6 Remote monitoring of the instrument

5.3. Scripting

Scripting support is essential for all control systems that provide fast creation of operating sequences by users.
The Container formally forbids scripting inside the wiring file not only to make it more maintainable but also to keep the scripting language exchangeable.
Adding new scripts using Inspire Modules in the same way as with all other modules has several advantages. The most evident advantage is a security issue. Wiring only the devices to the scripting modules that are to be manipulated by the script provides a means of information hiding and restricts the access to uncritical devices. One other advantage is the ability to transfer the script to other instruments since it is separated by the configuration code.
Open Inspire supports scripting in all languages that support the JSR 223 Scripting API [20]. This allows users to select his preferred scripting language from a list of over 25 scripting engines such as Python, Groovy, Ruby, Tcl, AWK, BeanShell or Java Script.
The preferred scripting language supported by Open Inspire is Groovy [21]. It is particularly suitable for instrument control as it combines the power of Java with the clear and easy scripting capabilities known from scripting such languages as Python.

5.4. Database

Persisting configuration and experiment data can be realized using data storage modules.
The OIDG proposes using Java standards as the Java Persistence API [22] to store and reload internal module data preferably in the Java DB [23].
For experimental data it is recommended to use the NeXus [24] support. The *NativeNeXus* module wraps the official NeXus API and provides reading and writing hdf4 [25], hdf5 [25] and XML files depending on NeXus extensions installed on the Application Container host.

5.5. Simulation

Experiment time is limited and valuable. Using simulations for preliminary tests, experiment planning or calculation of non-measurable data reduces measurement time, enhances quality and allows new types of experiments.

Developing a new beam line simulation framework is counterproductive because good simulation frameworks are already developed.
Open Inspire modules are available for simulations using the McStas [26] simulation package. This module supports starting simulations and reading back the results.
To be able to address the simulation in the same way as the real experiment the modules *McStasMotor*, *McStasDetector, McStasCounter* implement the standard interfaces in *DomainScience*.
The executable simulation, compiled by McStas for the experiment, needs to be available beforehand. Starting a scan submits the values of motors and counters to the simulation and reads back the detector data, which is accessible by the *McStasDetector* module. Running the same experiment with the real and virtual experiment is easy since both implement the same interfaces.

An alpha release of a Netbeans based rich client for the development of McStas simulations is available and will be integrated in future releases of Sunrise.

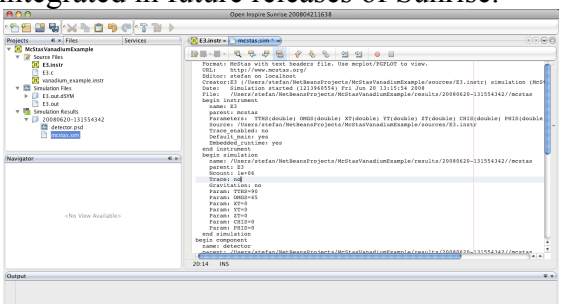

Figure 7 The alpha of the McStas simulation editor

Motors and counters can be simulated by a reproduction of the specific device behavior using a thread that simulates the succeeding of the actual value while reaching a given set point.

Simulating a camera picture can be done using the camera server that provides a simulation of the sample movement in front of the camera.

5.6. Calibration

Comparing real experiment data with simulated data assumes an experiment to be set up in the same way. Since simulated instruments always provide a geometrically perfect setup, real experiments are affected by positioning tolerances. The Calibration Modules can be used to setup the instrument in an accurate way by centering the sample on the table and positioning all neutron optics. A user interface monitors the camera picture and edge values and can be used to set edge detection parameters.

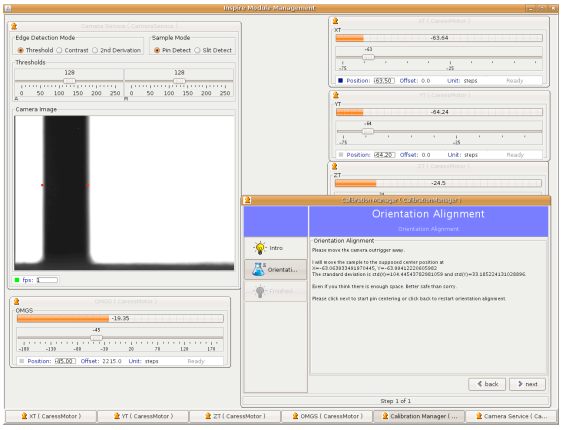

Figure 8 The calibration GUI in Inspire 2007

6. Setup and Development

With Open Inspire 2008 the development methodologies changed from a strict UML based top down modeling to a more agile community based approach. To assist this process, a new user-friendly build system, several web resources and development tools have been introduced.

6.1. Binaries and Resources

The first place to go for obtaining the Application Server, the Sunrise Client, official modules and development tools is http://www.openinspire.org.
The new project website provides binary distributions, modules, documentation, developer tools and community pages.
Official public access to the new resources is planned for January 2009.

6.2. Source Code

Open Inspire recently moved from Subversion to a Mercurial [27] repository at http://hg.openinspire.org.
Developers with a valid repository account can check in and out all sources of the Application Server, Sunrise Client official modules and development tools.

6.3. Build System

Open Inspire 2008 provides a new and easy to handle build system that is extended by several new features and automates as many steps as possible.

Building, cleaning and deploying the Application Server, Sunrise Client and modules do not require more than a recent Ant [28] installation and a command line. Dependencies between modules are now automatically resolved and compiled in their right dependency order. Necessary Ant extensions are compiled during the first build and extend Ant with Inspire specific features as sort algorithms or automatic downloading of missing modules from a central repository. To simplify the installation of preconfigured builds, ZIP-Distributions can be created.

6.4. Debugging

The Application Server and its modules can be debugged by connecting a JDPA[5] [29] compliant debugger to the applications JDI[6]. The preferred way for debugging and profiling the application is by using the Netbeans IDE.

All command line targets such as build, clean, deploy and additional targets like debug and profile are available from within the IDE.

A module called *WidgetTestcenter* can be used for visual debugging of modules and widgets. This module behaves like other modules but opens a server side desktop window that shows widgets for all connected modules. The *WidgetTestcenter* is also useful when a server side graphical access for configuration and monitoring is required in case of a missing client.

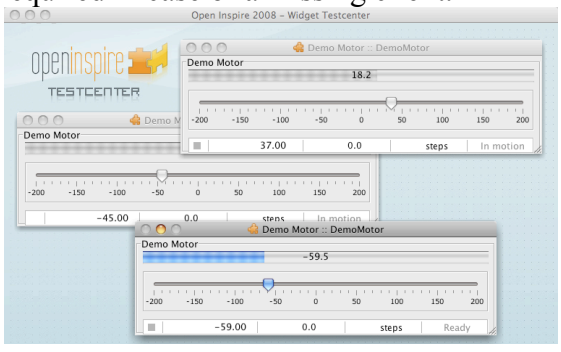

Figure 9 Debugging Motor Widgets in Inspire 2008

[5] Java Platform Debugger Architecture
[6] Java Debug Interface

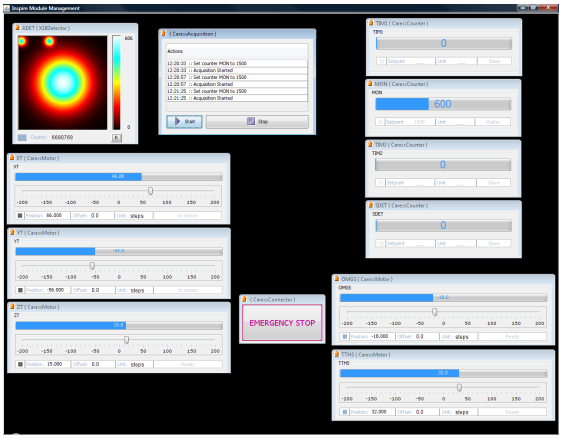

Figure 10 Running a Debug Scan in the Inspire 07

6.5. Development Tools

The standard development environment for the server, client and modules is the Netbeans IDE [30] although other IDEs can be used as well. Netbeans provides all necessary features such as access to all build targets, refactoring over all modules, debugging, profiling or syntax highlighting. Whenever a new IDE based OI development tool is available, it will be first published for Netbeans IDE.

Tools that are not based on an IDE can be downloaded or directly used online. One example on the website is the OIM generator that asks for meta-information such as name, description, icons, authors and dependencies and generates a preconfigured module in the form of a Zip-File for download.

7. Conclusion and Summary

The aim to provide an instrument software infrastructure with a maximum lifetime in a constantly changing environment has been reached. Open Inspire is ready to be integrated in existing environments and provides the glue between incompatible systems by bridging between legacy architectures and new developments. The software covers all fragments of a complete instrument system including hardware servers, scripting, monitoring, simulation, control and user interfaces. An instrument responsible may decide

whether to use a standalone setup or just filling gaps in existing systems.

Providing a durable system in a shifting environment assumes fast reactions on changes that can be achieved by a flexible component model. The usage of a central Application Container surrounded by services solves several design issues in an elegant way:
All modules are independent and can be replaced or refactored without affecting the overall system. Each module can be directly used in other applications since no internal bindings to the container or other modules are required. Wiring modules by references in a central container improves performance, reduces traffic and enables easy debugging and profiling. Centralized configuration and security is much better maintainable and it enables to implement aspect-oriented features such as a global logging service. Furthermore updates or upgrades to the container and modules can be guaranteed and handled in one place.

A changing environment also means changing users and teams. To provide a steep learning curve for new members, work units can be clearly separated and protected by means of information hiding.

Developers only need to know the desired interfaces when creating new modules or refactor existing code. Only knowledge about the specific work unit is required and parallel team development is easier. Tools for automated build and clean, a module generator and IDE based debugging and profiling is supported as well as deployment helpers.

Instrument Administrators can use and configure modules without knowing the internals, provide fast and flexible setups for new experiments and share wiring files with other administrators. Open Inspire Sunrise provides ways for graphical configuration and simplifies the usage in restricted networks by its tunneling capabilities. Modules in Sunrise can be installed on demand to provide a clearly arranged user interface.

Users can profit by fast setups for new experiments. Measurements can be monitored and problems can be corrected from different networks when the administrator permits it. Using the Inspire scripting module moreover allows writing scripts in the preferred scripting language such as Groovy, Python or Java Script.

8. Outlook

Currently all code of the Open Inspire Application Server, the OI Sunrise Client and official modules is in revision and prepared to be licensed under the terms of an Open Source license. This step is necessary for opening the Inspire source code repository to the public.
New manuals, tutorials and examples are in development for the new Open Inspire project website that will help users and developers starting with Open Inspire. Old tutorials and documentation will be revised for the new version. The redesigned website also provides downloads of recent versions and makes internal services available for the public. This includes the module generator, forums, repository- and module browser.

Work is also going into the Sunrise Client. Enhancements and bug fixes for the wiring editor, module activation, event handling and new module frontends are in progress.

Furthermore a complete refactoring of the online module repository is in progress to enable a better versioning of modules. This includes a new automatic dependency resolver for the container, the build system and the new online module browser.

Last but not least can be expected that new modules will be published and shared with the community. Parts of the new beta modules such as NeXus and McStas

support will be rewritten for better integration.

9. References

[1] Martin Fowler, 2004, Inversion of Control Containers and the Dependency Injection pattern
http://martinfowler.com/articles/injection.html
[2]Application Server
http://en.wikipedia.org/wiki/Application_server
[3] Loose coupling
http://en.wikipedia.org/wiki/Loose_coupling
[4] CORBA (Common Object Request Broker Architecture)
http://www.omg.org/corba/
[5] CARESS
http://www.hmi.de/bereiche/I/DE/systeme/caress.html
[6] SOAP (Simple Object Access Protocol)
http://www.w3.org/TR/soap/
[7] RMI (Remote Method Invocation)
http://java.sun.com/javase/technologies/core/basic/rmi/index.jsp
[8] Middleware
http://en.wikipedia.org/wiki/Middleware
[9] JEE, Java Platform Enterprise Edition
http://java.sun.com/javaee/
[10] Spring Framework
http://www.springframework.org/
[11] JAR, Java Archive File
http://en.wikipedia.org/wiki/JAR_file
[12] Java Annotations
http://java.sun.com/j2se/1.5.0/docs/guide/language/annotations.html
[13] Java Beans
http://en.wikipedia.org/wiki/JavaBeans
[14] Java System Tray functionality
http://java.sun.com/developer/technicalArticles/J2SE/Desktop/javase6/systemtray/
[15] Netbeans Rich Client Platform
http://platform.netbeans.org/
[16] VisAD (Visualization for Algorithm Development)
http://www.ssec.wisc.edu/~billh/visad.htm
[17] SSH Tunneling
http://www.ssh.com/support/documentation/online/ssh/winhelp/32/Tunneling_Explained.html
[18] The Cajo Framework
https://cajo.dev.java.net/
[19] GUM (Grand Unified Model), A.Götz, N.Hauser, 2004
http://lns00.psi.ch/nobugs2004/papers/paper00125.pdf
[20] JSR 223 (Scripting for the Java Platform)
http://jcp.org/en/jsr/detail?id=223
[21] Groovy, An agile dynamic language for the Java Platform
http://groovy.codehaus.org/
[22] JPA Java Persistance API (JSR 317)
http://jcp.org/en/jsr/detail?id=317
[23] JavaDB, Apache Derby Database
http://developers.sun.com/javadb/
[24] NeXus
http://www.nexusformat.org/Main_Page
[25] The HDF dataformat
http://www.hdfgroup.org/
[26] McStas – A neutron ray-trace simulation package
http://neutron.risoe.dk/
[27] Mercurial – A distributed version control system
http://www.selenic.com/mercurial/wiki/
[28] The Apache Ant Project
http://ant.apache.org/
[29] Java Platform Debugger Architecture
http://java.sun.com/javase/technologies/core/toolsapis/jpda/
[30] Netbeans IDE
http://www.netbeans.org/