OBJECT EXPLORER - A PLUGGABLE GENERIC TESTING AND DIAGNOSTIC TOOL

M. Kadunc[*], I. Križnar, M. Pleško, M. Šekoranja, G. Tkačik, I. Verstovšek
J. Stefan Institute and Cosylab Ltd., Slovenia

Abstract

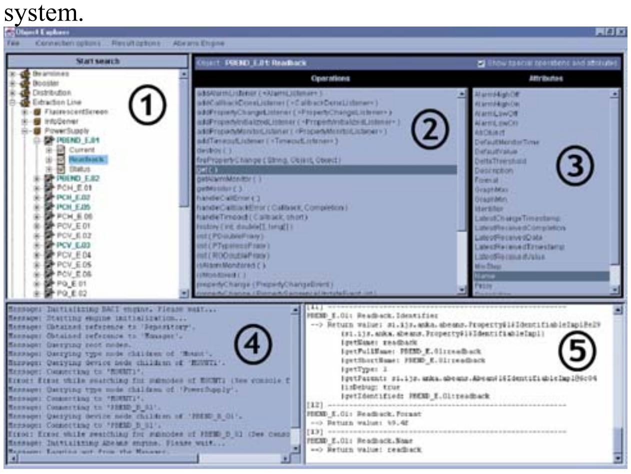
Object Explorer is a generic application that discovers at run time all devices, properties or channels that can be controlled, analyses them using CORBA and Java™ introspection tools and displays methods, fields, commands and other actions in a GUI. The user can select and invoke these actions, including entering arbitrary parameters. The application was first developed by the KGB group at JSI [1] for testing and debugging the ANKA control system. A completely new version has now been developed by Cosylab [2] for the ACS control system software and the Abeans API. The application is divided into two parts: The engine part is control system-specific and communicates with the underlying data or server layer, providing data for the GUI part and passing user requests to the control system. This layer is relatively thin and can easily be implemented for many server protocols such as TINE, EPICS etc. The GUI part is independent of the engine implementation and of the communication protocol used. It receives data from the engine and shows all the available information to the user. This article describes the functionality of the Object Explorer, some of the software designs used, and discusses some ideas which might be implemented in future releases of the application.

1 INTRODUCTION

When we were developing the control system for the ANKA accelerator[3,4], we needed a client tool to test our device servers. CORBA, which was used for communication, provided two useful generic services[5]:

- Interface Repository, a service that provides information about the objects that can be accessed on a certain network.
- DII (Dynamic Invocation Interface), a service that offers a generic way of manipulating objects, i.e. invoking methods and retrieving fields.

Using these principles we were able to create Object Explorer, an application that provides run-time information about a control system. It discovers device servers, displays lists of devices in a GUI and allows user to read the data on the devices, invoke all control system commands and create monitors. The application served its purpose well and almost all of the servers' functionality was tested with this tool. As we started to work on ACS[6], a new control system developed for the ALMA project in collaboration with ESO[7], we decided to rewrite the application. The old one was too tightly connected to the structure of the ANKA control system, to the implementation of CORBA used there, and was written in Java 1.1, which was long outdated. Modifying the existing application would take a lot of effort and little results, as we would have to do it again for our next project. The new application is flexible enough that enabling it for new control systems takes very little programming effort. It is also very generic so that it does not need to be modified for each change of the objects' hierarchy or their declaration. The ACS is now in the final development phase and Object Explorer has been the primary tool used for testing and debugging the control system.



```
Message: Initializing BACI engine. Please wait...
Message: Starting engine initialization...
Message: Obtained reference to 'Repository'.
Message: Obtained reference to 'Manager'.
Message: Querying root nodes.
Message: Querying type node children of 'Mount'.
Message: Querying device node children of 'MOUNT1'.
Message: Connecting to 'MOUNT1'.
Error: Error while searching for subnodes of MOUNT1 (See console f
Message: Querying type node children of 'PowerSupply'.
Message: Connecting to 'MOUNT1'.
Message: Connecting to 'PBEND_B_01'.
Message: Querying device node children of 'PBEND_B_01'.
Message: Connecting to 'PBEND_B_01'.
Error: Error while searching for subnodes of PBEND_B_01 (See conso
Message: Initializing Abeans engine. Please wait...
```

```
[11] -----------------------------------------------------------
PBEND_E.01: Readback.Identifier
 --> Return value: si.ijs.anka.abeans.Property$1$IdentifiableImpl8e29
        (si.ijs.anka.abeans.Property$1$IdentifiableImpl)
        |getName: readback
        |getFullName: PBEND_E.01:readback
        |getShortName: PBEND_E.01:readback
        |getType: 1
        |getParent: si.ijs.anka.abeans.Abean$1$IdentifiableImpl06c04
        |isDebug: true
        |getIdentifier: PBEND_E.01:readback
[12] -----------------------------------------------------------
PBEND_E.01: Readback.Format
 --> Return value: %9.4f
[13] -----------------------------------------------------------
PBEND_E.01: Readback.Name
 --> Return value: readback
```

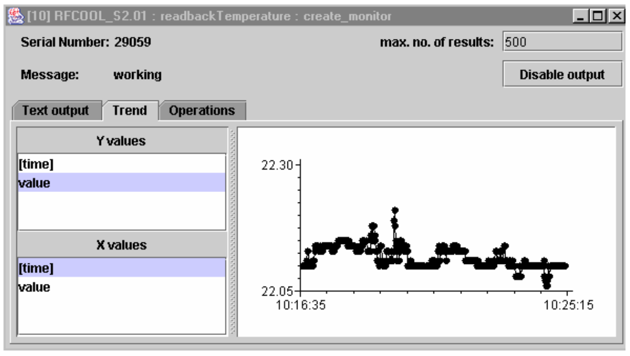
Figure 1: The main application window

2 WALKTROUGH

The user starts with selecting the system: the Object Explorer either connects to the ACS, communicating with MACI (Management and access control interface) supervisor service called Manager, or connects to the Abeans framework [8] as an ordinary Abeans application. The structure of the control system is displayed in a tree component (Figure 1/1). The system is first queried for the root containers, and as the user expands nodes in the tree, more queries are made to retrieve other containers and finally the control system objects. User can connect to these objects to get a list of all their supported members – operations (Figure 1/2) and attributes (Figure 1/3). The user can invoke operations, parameters that they need are entered into parameter entry fields that show in a dialog (Figure 3). User can enter primitive types such as numbers or strings and construct simple objects. If an operation is synchronous, the result is immediately printed into a text area at the bottom of the screen (Figure 1/5); otherwise a node (request node) is added to the tree as a child of the inspected object, specifying that an

*miha.kadunc@cosylab.com

operation on the object is waiting for a response. The node itself is an active object that can have its operations or attributes. Responses from asynchronous requests can be printed to the same text area, or displayed in separate windows, called Remote Response Windows (Figure 2). Each request has its own window where responses are displayed to the user. Optionally, the numeric values that are parts of a response are plotted in a trend chart. Response nodes' members are also displayed in a list and can be invoked from the window.


[10] RFCOOL_S2.01 : readbackTemperature : create_monitor

Serial Number: 29059

max. no. of results: 500

Message: working

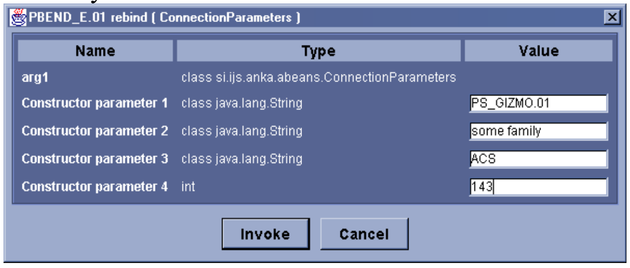
Figure 2: Remote response window displaying a trend chart

3 ARCHITECTURE

The Object Explorer was developed at the beginning of the ACS development and we had enough time to design the architecture of the application well and implement the application in a generic way, using all possible standards and services provided by Java™ and CORBA. We decided to separate the application into two parts – application GUI and device server access part (called the engine), and that we would connect these two parts as loosely as possible, thus making the implementations clearer, easier to maintain and extend.

3.1 Application GUI

The GUI part handles all the visual aspects of the application, as well as the lifecycles of the engine and other objects. It assumes that data are retrieved from remote servers and is optimized for remote communication – the data are only requested when a tree node is expanded, rather than loading the whole structure at start-up. Remote operations, such as method invocations, structure queries, etc. are run in separate threads, keeping the application responsive while the user is waiting for callbacks and making it more stable in case of communication errors and other exceptions. The application GUI is designed for expert users who know their control system well and understand every aspect of its functionality. It does not try to be user-friendly in a way that it would display only simple errors or relevant data, but prints out all the information it can obtain, regardless of their importance. The GUI part uses Java Introspection to unpack the data retrieved from the remote system. Human readable data is printed to the report area, more complex objects recursively queried for their public fields and a detailed tree-structured report is constructed. Introspection is also used to construct the parameter entry dialog (Figure 3). Parameter types are analyzed and appropriate GUI controls are constructed allowing the user to enter complex parameters such as structs, enums and arrays.


PBEND_E.01 rebind (ConnectionParameters)

Name	Type	Value
arg1	class si.ijs.anka.abeans.ConnectionParameters	
Constructor parameter 1	class java.lang.String	PS_GIZMO.01
Constructor parameter 2	class java.lang.String	some family
Constructor parameter 3	class java.lang.String	ACS
Constructor parameter 4	int	143

Figure 3: Parameter entry dialog – constructing a complex type

3.2 Engine

The only part of the Object Explorer that depends on the client-server communication architecture (e.g. BACI CORBA, Abeans, TINE, etc.) is called an engine. All engines implement a set of interfaces, needed by the GUI for data access. This layer is relatively thin and easy to implement for a given control system, as most of the usual problems are solved by the GUI part. Since the whole application is designed to be as generic as possible, the engine implementations should also allow for different types of objects to be analyzed, which is the most difficult part of an engine implementation, but pays off when using the application.

3.2.1 BACI engine

The BACI engine establishes a connection to the ACS' managers using CORBA. It queries the BACI Manager to obtain a list of device names, which are first displayed in the tree and later used to connect to remote entities - either BACI DOs (Distributed Objects) or BACI Properties. Furthermore, CORBA introspection (Interface Repository) is used to determine BACI compliance and identify design patterns, such as actions, static items etc. The engine converts the CORBA parameter types to standard Java types or predefined BACI structures, so that the GUI can display them and allow users to enter the parameters. Data returned by the control system is also unpacked and converted. BACI engine implements callbacks as CORBA DSI (Dynamic Skeleton Interface) server. This means that it is capable of unpacking any callback if its interface is present in the interface repository.

3.2.2 ABEANS engine

The Abeans engine connects to the Abeans API, which is running in the same virtual machine as the Object Explorer. It registers to the framework as a normal Abeans application and uses the Abeans ServiceBean as a starting point. It queries the ServiceBean for domains, types, and names of devices that Abeans can access. Device beans are then instantiated and initialized so that they connect to the control system. Java introspection is used to analyze them and to retrieve all their methods and fields. Some Java design patterns, such as registration of event listeners, are identified. Users would have difficulties constructing event listeners, which would provide meaningful data about the events, during runtime, so the Abeans engine hides these complex parameters from the user and constructs default implementations of event listeners, which have in most cases proved sufficient. Abeans design patterns are also identified – BACI properties are handled separately as nodes in the tree, while other object fields are only displayed as attributes; Abeans monitors are displayed as request nodes - children of the object being monitored, and can be controlled from within the Object Explorer.

4 CONCEPTS

Generic tools are very powerful, as they instantly reflect any changes in the control system and need no modification when new functionality or new devices are added. To ensure this, introspection was widely used throughout the application. Introspection is the ability to obtain information about a class or an interface without having static (linked) access to the interface declaration. With introspection, one can get all the data about an interface (methods, method parameters, return types, fields, etc.) only by providing its name. Interfaces can be very wide and allow all possible parameters and return types, while most of the control systems and even most of everyday software is built according to some rules or contracts that software programmers must follow (design patterns). A simple example of design patterns would be Java Beans with property getters, setters and event listeners. Even if we get method signatures from some introspection service such as CORBA Interface Repository or Java Introspection, we have to analyze them to identify design patterns that are common to the inspected system. BACI engine, for example recognizes several design patterns of the BACI model:

- Property – property accessor method has a special signature
- Asynchronous action – one of the method parameters is of callback type
- Subscription (used in publisher – subscription design pattern) - the method returns an object of special type.

When BACI engine identifies these design patterns, it can process the operations and attributes accordingly, i.e. asks for the relevant parameters, assumes the returned object's persistence etc. Introspection alone would not provide this data and all the methods would be treated equally. With these second-level introspection capabilities, Object Explorer is not only useful to test whether the device servers produce any errors or device functions return the right values, but also to test the design of the device servers themselves. The Object Explorer compares its built-in design patterns to the actual objects and automatically complains if the objects do not follow the standards. If such tool would not be available, one would have to go through the device server code or even database and check for compliance.

5 PLANS FOR FUTURE RELEASES

Next version of the Object Explorer will be written using all the new features of Java 1.4 and will use many of the components and services developed by Cosylab for CosyBeans. By using the launcher framework of the CosyBeans library, the application will be able to run remotely using Java WebStart technology, as an applet from a web browser, as an internal frame in a desktop with other CosyBeans applications, or as a stand-alone application.

The GUI part of the application will change significantly – the appearance will be much more customizable, allowing the users to choose between different types of data representations and to create their own application layout. Data will be better organized, each object returned by the underlying layer as a result of an action will be an entity of its own and the users will be able to inspect each object to get the information they need. Users will have more freedom when entering parameters – they will be able to take an object they obtained from the control system, modify it, and pass it as a parameter to an operation. They will also be able to construct an arbitrary Java object, set its fields, invoke its methods and use it as a parameter when invoking remote operations. The application is intended to have a clipboard for easier manipulation of the most commonly used objects, an engine for checking the classpath for type hierarchy and interface implementers, improved error handling with detailed exception information, better logging capabilities etc. We are studying ways of dynamically defining and loading classes that would implement certain interfaces, if no appropriate implementers could be found in the classpath, but this is a rather far-fetched idea that will probably be very difficult to implement.

The engines will probably not change drastically, but some other software layers, such as the new Abeans Release 3 or TINE, will be supported.

6 REFERENCES

[1] http://kgb.ijs.si/
[2] http://www.cosylab.com/
[3] http://www.anka-online.de/
[4] J. Dovč et al., "ANKA Control System Takes Control", PCaPAC 2000, DESY, Hamburg, Germany
[5] G. Milčinski et al., "Experiences With Advanced CORBA Services", ICALEPCS 2001, San Jose, CA, USA
[6] G. Chiozzi et al., "Common Software for the ALMA Project", ICALEPCS 2001, San Jose, CA, USA
[7] http://www.eso.org/
[8] G. Tkačik et al., "Java Beans of Accelerator Devices for Rapid Application Development", PCaPAC 99, Tsukuba, Japan