ABEANS: APPLICATION DEVELOPMENT FRAMEWORK FOR JAVA

I. Verstovsek, M. Kadunc, J. Kamenik, I. Kriznar, G. Pajor, M. Plesko, A. Pucelj, M. Sekoranja, G. Tkacik, D. Vitas, Cosylab, Ljubljana, Slovenia

Abstract

Abeans is the next generation of Cosylab's Java based client framework for building control system applications. It has been ported, plugged to such different CS as those of DESY (TINE), SNS (EPICS) and GSI. At the moment we are working on the installations for ALMA project by ESO and for Data Acquisition group at JLab. Abeans consists of two parts, Abeans for control and CosyBeans. Abeans for control provide application services and the mechanisms that allow simple implementation of data flow between the local application and the remote control system. This task is realized in two layers. Firstly, Abeans define a model that is a layer of Java Beans components that represents controlled objects. Secondly, Abeans define a plug which is a driver layer, specific to a given model and an underlying communication system. In addition, Abeans provide several useful services: logging, exception handling, configuration and data resource loaders, authentication, and policy management. CosyBeans provide clear and consistent visualization of dynamic data with standardized presentation of alarms, monitors and connection status. In this article we present the basic concepts of Abeans and latest developments, such as the BACI model, developed for ESO, the possibility to write CosyBeans GUI components as JADE agents for JLab and AbeansDirectory, an Abeans service that is an implementation of JNDI DirContext. We continue by outlining some use cases where we show how these technical solutions help solve concrete problems the application programmer is facing in the real life. We will look at a generic application Object Explorer that is able to display and modify any value that is accessible by a given control system. This means that the same application can be used to explore TINE, EPICS or ACS CORBA based system. Second example will be a specialized panel that displays and controls values of a remote physical device. We will show that by using Abeans, the application developer just needs to take care of the user interface – by means of CosyBeans graphical displayers, he can construct a fully functional application in a visual editor without writing a single line of code, the lifecycle of remote entities and the installation of services is performed by the Abeans. As a third example, we will show how to make use of the Abeans when developing a complex application for machine physics that interacts with a large number of physical devices and depends heavily on the services provided by the framework.

INTRODUCTION

We will present what Abeans are today through a historical perspective, describing their evolution from the first release to the current front runner – Abeans Release 3. In section 3 we continue by explaining the basic concepts of Abeans: models, plugs, services and CosyBeans. Section 4 provides a quick glance into a variety of control systems and the benefits and challenges one is faced with when using them.

Section 5 shows how lugging Abeans to a given CS facilitated finding a particular solution for a given issue and in parallel explains some additional Abeans concepts. In section 6 we summarize and try to show how Abeans can provide general solutions to specific problems. We conclude by outlining a path for further evolution of Abeans.

A BRIEF HISTORY

The story of Abeans began in 1997 when a group of students led by Mark Plesko started the development of a control system for the accelerator ANKA in Germany based on CORBA middleware [1]. As the students mostly could not work on a regular basis, tasks of sufficiently small size had to be made that required well-defined and stable interfaces, both in hardware and software. A good design was crucial insofar as it partitioned the project into small and manageable tasks, which did not require a lot of extra work to assemble in the end. It was also important that the assembly of the pieces would not pose extra work [2].

This line of reasoning lead to horizontal division of tasks and responsibilities: each team member needed to master and implement a specific module, instead of being responsible for the control of a particular kind of equipment, which would correspond to vertical division. Such modules were generic and thus not related to specific equipment in order to hide complex details from application programmers. From this arose the idea of a Java-CORBA wrapper for the developers of Java applications that would hide the details of CORBA. The wrapper was very thin, exactly replicating the CORBA IDL interface, but without the CORBA syntax. There was almost no additional functionality. This was the first release of Abeans.

The concept to have a wrapper for communication between the remote control system and the application developer was transferred to other systems as well, such as RIKEN and ESO. In order to achieve this, the concept of a plug was introduced: a plug is the part that is responsible for translation of requests originating in the GUI to a communication protocol used by the control system and for the interpretation of responses produced by the remote system. The purpose of the plug was to keep the interaction with GUI constant, while allowing for communication protocol to change. Plugs for several different CORBA systems were developed. This was the second release of Abeans.

In addition, the second release made more extensive use of Java Beans features and declared a consistent set of events and properties that facilitated RAD development. Features central to some control systems (but not present in the others), such as packed monitoring, access to the naming service and remote logging, were integrated as late additions.

Thus, hiding the implementation specifics of the remote layer is not everything there is to the story and moreover, late additions were difficult to abstract into communication-system-independent concepts. To us these facts indicated the need for a new Abeans design, named Abeans Release 3 (simply referred as Abeans from now on). The main concepts of the new design are discussed in the next section.

BASIC CONCEPTS OF ABEANS

Given some complex software system, let us say a distributed system or for example a database, Abeans can firstly be used to build a *model* of the complex controlled system, secondly to build a *plug* for communication with the complex system and finally to organize *services* not related directly to the complex system, but to task of application building. In addition, a separate part of Abeans is dedicated to the visualization of remote data – this part are the CosyBeans.

Model

Abeans provide the building blocks for constructing an object-oriented representation of some complex system. A model is a set of Java classes that represent the components of a given system, and those classes are common to all systems from a certain functional / problem area. In other words, in order to control physical devices such as power supplies, vacuum pumps, etc., Abeans declare, for each physical device, an object instance in the OOP sense, and define the lifecycle, containment and relation to other services.

The fist model that was implemented was the **Channel model** – the remote data is accessible through separate channel objects and is well suited to model systems that are flat (not organized hierarchically), like for example in EPICS or TINE protocols. There is one model for all channel based systems (not a separate model for TINE channel, and EPICS channel).

The second is the **BACI model** – here a set of properties can be part of a hierarchical entity, for example a *device*, that mirrors the logical structure of physical devices, like power supplies. The main point is that the device is a Java object. This model was primarily developed for ESO and uses ACS CORBA plug below it.

Plug

The concept of a plug was already mentioned in the historical introduction. Abeans usually communicate with some already existing software that offers access to the data. A plug is that part of the Abeans system that is responsible for translation of requests originating in Abeans (in GUI, services, etc.) to a communication protocol used by the controlled system.

So far, we have implemented plugs for TINE protocol for DESY at Hamburg, EPICS plug that communicates to the JCA library, UFC plug for the protocol used at GSI, and ACS CORBA plug for ESO.

Service

Although two Abeans applications that use two models for the control of two software systems differ in their basic purpose, they still contain a lot of shared functionality. Error reporting, logging, resource loading, configuration management and similar tasks can be delegated to a body of shared services, which is implemented once and for all. This approach reduces the amount of coding and guarantees consistent behavior, look and feel and similar functionality across all applications developed with the Abeans framework.

So far we have implemented the following services: configuration service, data resource service, debug service, exception handler service, report service and thread pooling service.

Abeans also offer standard interfaces for access to distributed services (i.e. services provided on remote machines), such as distributed naming service, or a distributed archive. We used existing technology where possible: for example, access to object directories (such as TINE Name server or ACS Manager) is done through standard JNDI (Java Naming and Directory Interface). Consequently, it is possible to develop a directory browsing Abeans application, which will run on all Abeans plugs – this is the Object Explorer. Similarly, distributed archive service can access archive data in remote archive servers. Archive reader application developed for DESY uses this service of Abeans. However, since the same service will be implemented for EPICS as well, we will be able to reuse the same application to access EPICS archives.

ISSUES RELATED TO VARIOUS CS

The purpose of this section is to provide some examples of control systems we have experience with and to present some specific challenges one faces when using them.

SNS, Diamond (EPICS): EPICS has a flat structure of its database. The central entity is the Process Variable (PV). The only way to define a functional container for PVs, i.e. a device, is to use a naming convention or to provide additional data stored in some sort of a database. Both approaches require the application developer to look for additional sources for the data, reducing transparency. *Is there a way to provide hierarchical data to the application developer so that s/he would not have to know and explicitly search for additional information about the hierarchical structure of the control system?*

In addition, the names of the PVs in EPICS applications must be hard-coded, there is no general way to browse for all the channels on the network.

ESO, ANKA (CORBA): *Using CORBA might seem complicated for a beginner or an inexperienced programmer*, the learning curve to understand the stanzas (initializing ORB and POA) required to get a CORBA based client application to run is quite steep.

On the other hand, CORBA provides plenty of CORBA services that can be used on the device server layer (interface repository, naming service, etc), but *lacks components for visualization one could use in the client applications.*

GSI (UFC, ACS): The GSI accelerators are operated by a control system using VME computers on the device server level and OpenVMS workstations for the operator interface. Both layers communicate by a proprietary in-house protocol which is, on the client side, implemented for OpenVMS only. In addition, there is also one server which translates UFC protocol to TCP/IP based communication. After 15 years of operation, GSI is in the process of modernizing its control system in hardware and software.

The existing GSI control system cannot be completely replaced in a short time, to make a smooth transition, the GSI control system and ACS, the new control system candidate [3, 4] have to be supported in parallel during the transition period, both on the device server and the client application level – client applications must know how to connect both to new and to old devices. *Is it possible to write applications that will talk to both control systems at the same time? Will it be possible to use these applications after the transition period is over without much modification to their code?*

JLab-DAQ (JADE, EPICS): In Jefferson Lab, a new experimental hall will be built (Hall D). The possibility to use some other CS than EPICS or CORBA to control the experimental hall is being considered, perhaps the Channel Access protocol without EPICS running on top of it or JADE, an agent based protocol. However, Hall D would still need access to some EPICS-based devices, so it would be interesting to use *a facade to hide EPICS, and to have client applications that are able to connect to two control systems at the time.*

SNS (XAL): In Spallation Neutron Source, XAL – a layer for developing machine physics applications – is being developed on top of EPICS [5]. In parallel, SNS wants to have *generic applications for oversight of the whole control system. As well, SNS is interested to use GUI widgets and wants them to connect to the remote EPICS layer.*

ADDRESSING THE ISSUES

In this section, we will try to explain how each issue from the previous section was addressed using Abeans. Along the way, we will show additional concepts and features of Abeans.

Introducing hierarchies to EPICS

Abeans directory is a service that allows the application developer to do a lookup of the structure of names of remote objects. For example, available EPICS channels may be arranged into a sort of naming hierarchy based on some criteria (physical layout of the remote system, etc). Abeans plug for EPICS is responsible for providing information to the Abeans directory – the plug can obtain hierarchical information from the naming convention, relational database, some XML file, etc.

Abeans directory allows the developer to browse all available names for this plug, as if they were placed in a tree. The implementation of Abeans directory is in accordance with the JNDI specification which is a Java core platform. The added value for the developer is an intuitive and standard way to access the hierarchical data and not having to worry where the data comes from. Furthermore, the tree already has a GUI component that is able to represent it, the Cosy Navigator, which is a part of CosyBeans, and is displayed on Figure 1.

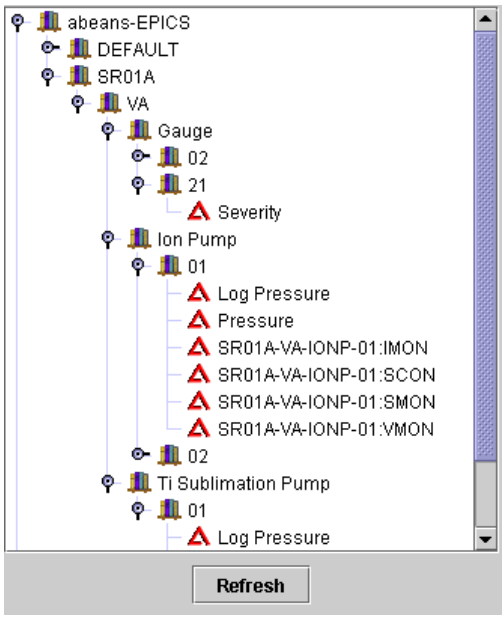

Figure 1: Cosy Navigator, a GUI component used to display and manipulate hierarchical device data.

Once the plug fills up the JNDI tree, the application can browse it and get all the channel names without having to know them in advance, thus eliminating the need for hard coding the names in the client application.

The directory actually contains meta-information about a named entity that can be accessed without actually making a connection to that entity. [6, 7].

Abeans can bring OO devices to EPICS

The discussion above refers to obtaining the references to channel names and constructing the channel objects, all the time using the Channel model. However, by having access to hierarchical data, it is possible to develop the idea further – to construct a logical representation of the device on the OO level. For example, one can have an object of type `IonPump` and access its properties (PVs) without having to explicitly connect to them or to know their names – once the user constructs the `IonPump`

object, s/he can retrieve its pressure property solely by invoking `ionPump.getPressure().getValue()`.

In addition to properties, a device can also define actions: functions that can be invoked directly on the device: instead of having a POWER property set to ON and OFF enumeration (effectively, writing 0 and 1 into it), actions can be invoked as calls to the functions on the `IonPump` device: `ionPump.on()` and `ionPump.off()`. Internally, the device object should map these calls into setting the appropriate values on the POWER PV.

The hierarchical model for EPICS will be developed in collaboration with Diamond [8]. When implementing the hierarchical model for EPICS it will be possible to reuse in great extent an existing implementation of a hierarchical model for Abeans – the BACI model that is used for ACS CORBA control system. However, there are some differences between the two cases: In the case of the model for ESO, device objects are static – they are Java classes generated from the CORBA IDL files (The IDL files define the structure of the device object). In the case of EPICS, the definition of what is a device (what properties and actions a given property contains) should be read from the directory.

Adding value to CORBA

As described in our brief historical introduction, the reason for the creation of Abeans was to hide the internal workings of CORBA from the developers of client applications for the control system for ANKA. Abeans were very successful in this respect. For example, the author of this article was able to develop the ramping application which accessed more or less all of the power supplies in the system without knowing at all what CORBA is all about! Even if one is proficient in CORBA, using Abeans still adds value – the developer has to write the stanza to connect to CORBA only once (in the plug), therefore reducing the need to copy and paste between different applications, as a consequence this increases the quality and maintainability of the code.

CORBA implementations are rich in providing additional services that are extremely useful for developers of device servers. These services include the interface repository, naming service, telecom log service, notification service, etc. By applying clean device patterns, ACS implements several system-wide services on top of these, such as logging, exception handling, connection management, etc. Using Abeans, it is easy to encapsulate these services and provide them to application developers as remote services, as described in section 3.3. Abeans user does not even realize that his "familiar" Abeans services in reality talk to underlying CORBA services.

Abeans enable a smooth upgrade of the GSI CS

In November 2002, a proof of principle Abeans plug was created for GSI. The plug connected Abeans to the UFC layer. Abeans plug was connected to a gateway server written in Java that was accessible via the TCP/IP protocol. The next step is to define an ACS based CORBA-object on the device server level. During the transitional period the applications should also be able to communicate with the old devices running on UFC.

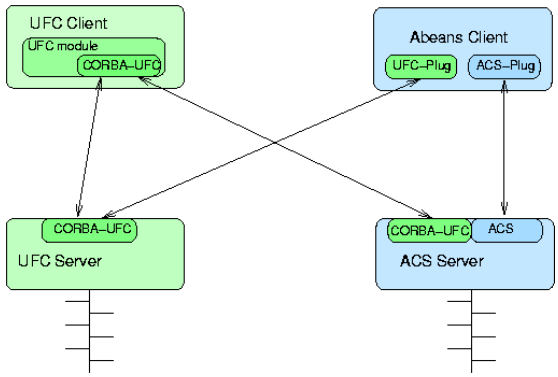

Figure 2: Abeans Client (top right) uses two plugs simultaneously, the UFC and ACS plug. Taken from [4].

One possible approach that enables a sequential upgrade is to use the following feature of Abeans: In the same application, it is possible to instantiate two plugs and therefore connects to two control systems at the same time. These plugs are the CORBA-UFC plug for old devices and ACS CORBA plug for new devices. During the transition period, more and more devices will be running on ACS. On the client level, changing the device to connect to a different plug is a matter of changing one entry in the configuration file without changing the code at all, which greatly simplifies the transition.

In JLab-DAQ Abeans are clients and servers

A very interesting possibility is to use Abeans for all the applications in Hall D. Because of the agent oriented approach of JADE, the main requirement in this case would be that Abeans applications communicate with each another – Abeans would be used as servers. This possibility is foreseen in the Abeans architecture, in fact, Abeans are currently used as server applications in the project for DESY, Germany – in this implementation SOAP protocol is used. Another benefit of using Abeans for all applications in Hall D is that the same Abeans application can be used to connect to multiple layers of the control system by means of different Abeans plugs, making the infrastructure very convenient both for the end users and for the application developers.

Abeans would be used to hide EPICS as well as for connection to JADE and to use them to control and configure the experiments. In fact, experiment control is not much different than hardware control – one always controls physical objects which have a state that exists outside of your local Java application.

In SNS Abeans are a GUI and services library

For the SNS project, XAL was developed to model the accelerator from the point of view of a physicist. In the case of SNS, Abeans are primarily used as a library – developers take from Abeans what they need and

incorporate it into the applications. In this case, the CosyBeans GUI widgets are the ones that are used most often, on figure 3 one of them, the Wheelswitch is displayed. The others are Gauger, Ledder, Object Table, Piper, Time selector, etc.

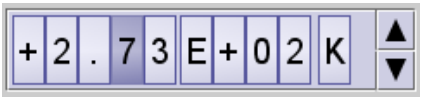

Figure 3: Wheelswitch, one of many GUI components of CosyBeans.

WHAT DO WE LEARN FROM THIS?

In previous sections we have taken a look at a wide set of issues one faces when dealing with different control systems. At a first glance, the problems did not seem to have much in common, and the solutions that were implemented with Abeans could have also been implemented in some other manner. However, at the end of the day, we have seen that Abeans addressed these issues in a unified way, by applying the basic concepts described in section 3.[0]

If we are allowed to oversimplify a bit, the task of writing control system boils down to a data flow problem where we must:

a) define and transform between possible formats of data (e.g. a double number that is annotated with characteristics),
b) define a (possibly hierarchical) object-oriented model of physical devices that provide access to data sources,
c) in step b) focus on the fact that modelling is functional and does not reflect specific HW or engineering choices.

Because functionality is shared between different machines, while HW and engineering details are not, and we take care to apply step c) consistently, Abeans provide an uniform abstraction which constitutes their added value in the field of control systems.

FURTHER EVOLUTION

Abeans were developed in order to solve problems that one faces when writing applications for control systems. However, a large number of application development issues are shared between applications in various problem areas, say for banking or any other application that uses distributed computing. Therefore, the same problems that Abeans solve are also being solved by some other, much wider frameworks.

One of the main priorities for further Abeans evolution is to make the integration of existing 3rd party software easier. In this manner it would be possible to integrate Abeans with a given control system not in a manner that the entire system is wrapped by Abeans (in some cases this might be an overkill), but to use Abeans in line with the existing system. One example where this is already the case is SNS where Abeans are used as a library. In this case it is possible to use Abeans' services and other useful “goodies” and at the same time to use the existing access to the system in the same application.

CONCLUSION

In the article we have presented how Abeans evolved over time and what have become today – they can be used either as a self-contained client application writing framework, or as a GUI, communication or services library. Recently, significant amount of time was devoted to write a bunch of extra documentation, also strict versioning was applied to Abeans code.

In general, we moved to a higher quality development process. The distributions can be downloaded from the Abeans homepage [9]. We believe, and a bunch of installations all over the world show, that Abeans can add value to both the application developers and end users of the control systems.

REFERENCES

[1] I. Kriznar et al, The Upgrade of the ANKA Control System to ACS (Advanced Control System), ICALEPCS 2003, Gyeongju, Korea, October 2003.
[2] M. Plesko et al, A Guerrilla Approach to Control System Development, ICALEPCS 2001, San Jose, California, USA, October 2001.
[3] G. Chiozzi et al, The ALMA Common Software: Status and Developments, ICALEPCS 2003, Gyeongju, Korea, October 2003.
[4] K. Höppner et al, Integration of Abeans and ACS in the GSI Controls Environment, ICALEPCS 2003, Gyeongju, Korea, October 2003.
[5] J. Galambos, et al, XAL Application Programming Framework, ICALEPCS 2003, Gyeongju, Korea, October 2003.
[6] M. Plesko et al, Where and What Exactly in “Knowledge” in Control Systems, ICALEPCS 2003, Gyeongju, Korea, October 2003.
[7] G. Tkacik, A Reflection on Introspection, ICALEPCS 2003, Gyeongju, Korea, October 2003.
[8] M. Heron et al, Diamond Light Source: Current Status and Developments, ICALEPCS 2003, Gyeongju, Korea, October 2003.
[9] http://abeans.cosylab.com