

NEW FEATURES FOR NEW APPLICATIONS WITH ABEANS 3.1

Andrej Košmrlj*, Igor Križnar
Jožef Stefan Institute and Cosylab Limited, Ljubljana, Slovenia

Abstract

Abeans are Java-based client framework for building control system applications. Cosylab has set as its primary design goal the ability to adapt them for a wide range of underlying architectures. By relying heavily on object oriented modelling, we have modularized them vertically into services, such as logging, exception handling, configuration and resource management; and horizontally into layers, such as plug layer, modelling layer and presentation layer. Portable generic applications as well as deployments in ANKA, ALMA, GSI, DESY, Diamond and SNS demonstrate that the basic premises of the design were sound.

In this article we discuss, using a specific example of Control Desk application (a generic table application) developed originally for Diamond, the main features available by the generic nature of Abeans. We address the basic question of how to trade flexibility for performance, and are careful to distinguish various kinds of overhead (one-time initialization, memory footprint, CPU consumption etc.). Strategies used in Abeans to improve performance of Java are presented.

INTRODUCTION

We will present what Abeans [1, 2] are today starting with explanation of the basic concepts of Abeans: models, plugs, services and CosyBeans. We will continue by providing a quick glance into the benefits and challenges one is faced with when using EPICS control system and then show how plugging Abeans to EPICS CS facilitated finding a particular solution for a given issue. We conclude by addressing the basic question of how to trade flexibility for performance and presenting strategies used in Abeans to improve performance of Java.

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 life cycle, containment and relation to other services.

The first 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

A plug is that part of the Abeans system 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.

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. We have also implemented SOAP plug for DESY that communicates via SOAP protocol to Abeans web service that communicates further through another plug. This plug was done as a research process and is currently not used in any deployed application.

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 functional-

* andrej.kosmrlj@cosylab.com

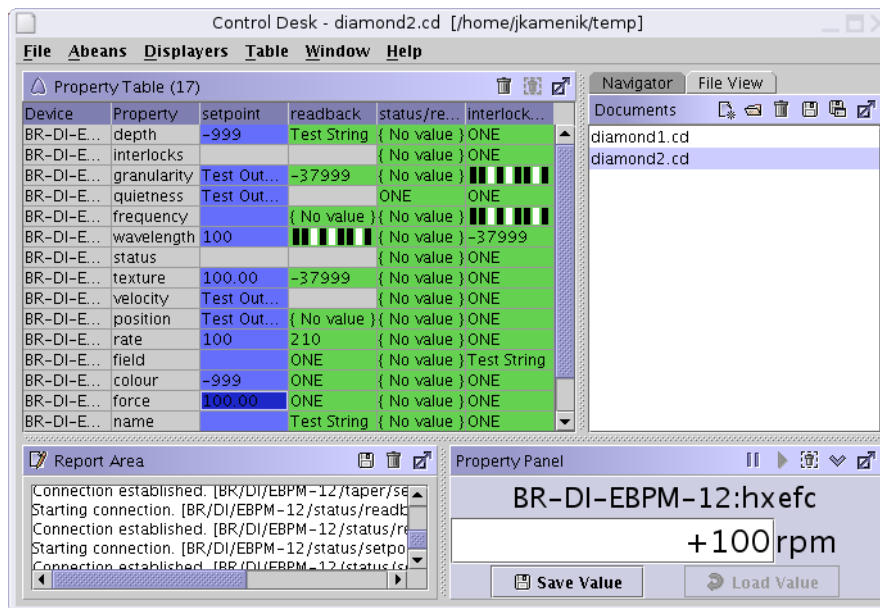


Figure 1: Control Desk application.

ity. 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 behaviour, 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 technologies 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.

CosyBeans

Where the path of the data item sent from the control system ends in Abeans, it continues with CosyBeans. CosyBeans are basically a data visualization library. As such, they consist of GUI components for display of single values of different types (doubles, bit-patterns, etc.), and for display of multiple data items (charts, tables and so on). To

couple the GUI components of CosyBeans – which can be used as standalone GUI components without any reference to Abeans libraries or other libraries – to their data sources (in our case Abeans), we provide Adapters. *Adapters* are thin pieces of software that connect, on one hand, to the Abeans models and on the other hand, to GUI *Displayers* that take care of actual data rendition. In general, Adapters could connect, instead of Abeans, to some other data source as well.

In addition to control system specific displayers and adapters, we also provide a bunch of stand-alone components for writing applications with Swing. These include enhanced swing components like button, text field, number field, table, active tree, etc. with additional features and performance improvements relevant to physicists. Some of the components were also developed from scratch, for example Spike chart, a high-speed chart built on the experience gained from long-time experience of our group with Java, About dialog, panels for reporting messages and exceptions and the like.

ISSUES RELATED TO EPICS CONTROL SYSTEM

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.

ADDRESSING THE ISSUES FOR CONTROL DESK APPLICATION

In this section, we will try to explain how each issue from the previous section was addressed using Abeans in Control Desk Application¹. This is basically a generic table application developed originally for Diamond that has several possible views. It is able to interpret the EPICS channels in a flat way, or in a hierarchical way by grouping them into devices. In the first case, each row represents a channel, and each column represents a (optionally specifiable) field of that channel. In the second case, each row represents a device, and each column displays a "default" field of a channel constituting the device.

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 obtains hierarchical information from the naming convention defined in some XML file.

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 2.

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. [3, 4].

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

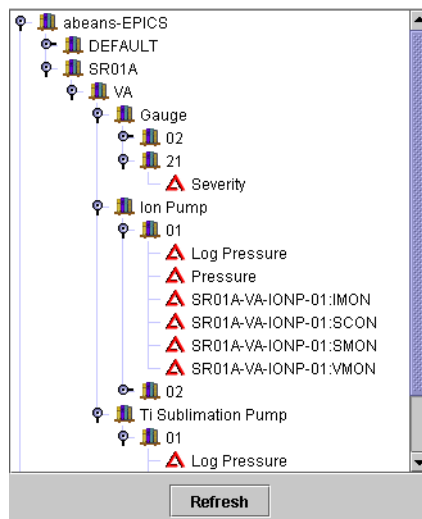


Figure 2: Cosy Navigator, a GUI component used to display and manipulate hierarchical device data.

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.

In Control Desk application we didn't go all the way to the Device model, but we made some intermediate step. In Abeans we introduced `PropertyGroup` which is a logical representation of the device and can also define actions via naming convention defined in some XML file. To keep `PropertyGroup` as generic as possible value of the device's properties is obtained like `ionPump.getProperty("pressure").getValue()` and actions are invoked like `ionPump.executeCommand("on")`, where the application can browse the JNDI tree to get all available properties and actions.

STRATEGIES USED FOR IMPROVING PERFORMANCE

In this section we will briefly describe what performance problems arose during development of Control Desk application and how did we cope with them.

¹see fig. 1

Memory footprint

Typical EPICS control system for a particle accelerator defines a large number of devices and their properties, but many of these devices are structurally the same (e.g. several ion pumps, magnets, power supplies, etc.). Each device is characterised with a modelling element descriptor in Abeans directory which describes meta information: names of the channels, commands and application attributes. It is easy to see that common meta descriptor for all structurally the same devices significantly reduces memory footprint. This is easily achieved if we have very strict naming convention (e.g. names of structurally the same devices and their properties differs only in prefix), but Abeans directory can handle even arbitrary names of devices, because there is a mapping between name of device and its descriptor. EPICS plug takes care that structural description of devices read from XML file is correctly represented in Abeans directory. XML structure already helps EPICS plug a lot, because we first define appropriate types (e.g. ion pump, magnet, power supply, etc.) with the whole structure. When we finally declare some instances of devices, we can define only its name and type.

Trading memory for speed

Abeans allows creation of several channels that map to the same PV in EPICS database. In this case plug creates a single PV that stores in cache all channel characteristics² (e.g. maximum, minimum, etc.) that are assumed to be constant all the time. Next time one of this characteristics is requested its value will be returned immediately and there will be no remote investigation to the EPICS database.

For a particular PV we must know its record type, so that a correct displayer will be used to present it. If we don't know the type of a PV, than we must synchronously connect to the PV and query for it. The solution to that problem is that we can specify type of the PV already in the XML file. For every PV we store in memory additional String but this improves speed up to 5 times.

Further speed optimizations

When coding, synchronous requests are much easier to implement, but it turned out that asynchronous requests are much more efficient, when we want to accomplish several requests at the same time. This is especially true for GUI adapters: they practically have no choice, they must use asynchronous requests because GUI must be responsive all the time. One must be aware that asynchronous calls should not significantly increase number of threads. If we have many threads, CPU consumption will drastically increase and CPU will spend most of the time switching between running threads instead of doing some useful job. JCA [6] interface for communication with EPICS database contains only asynchronous calls, which means that the

plug has no need to create any additional threads for asynchronous requests. This also means that plug must create a workaround for synchronous requests: this is obtained by standard Java `synchronized`, `wait` and `notify` methods. It turns out that Channel Access for Java (CAJ) [7] implementation of JCA interface is much more efficient than standard JCA implementation.

Further optimization is grouping requests as much as possible, especially for channel characteristics, because JCA interface contains methods for grouping remote calls, which turns out to be more efficient. Grouping requests for channel characteristics is important because we can get almost all relevant PV fields with a single call to JCA.

CONCLUSION

In the article we have presented how generic nature of Abeans can be used to address issues related to EPICS control system. This was used also in Control Desk application. Some features that were added to Abeans for EPICS, can be used also for other control systems. Abeans directory is already widely used in many applications, PropertyGroup can also be used for TINE and other similar control systems.

We introduced several strategies that were used to improve performance of the application, but we must continue doing so. At the moment we have problems during massive connection requests. Control Desk application can handle up to about 500 channels connection requests in one session. Most of our improvements were on the side of the plug. Now we need to find some improvements inside Abeans themselves and recently some minor improvements were already done. For example, the table application for APS Booster project can already handle up to about 2000 channels. Maybe the recent evolution of Java Virtual Machine [5] will be useful on our path.

REFERENCES

- [1] I. Verstovsek, et al, "The New Abeans and CosyBeans: Cutting Edge Application and User Interface Framework", PCaPAC 2002, Frascati, Italy, October 2002
- [2] I. Verstovsek, et al, "Abeans: Application Development Framework for Java", ICALEPCS 2003, Gyeongju, Korea, October 2003
- [3] M. Plesko, et al, "Where and What Exactly in "Knowledge" in Control Systems", ICALEPCS 2003, Gyeongju, Korea, October 2003
- [4] G. Tkacik, et al, "A Reflection on Introspection", ICALEPCS 2003, Gyeongju, Korea, October 2003
- [5] G. Tkacik, "Generic Types in Java: Abeans<T> Specifically for You, Mr T?", PCaPAC 2005, Sokendai, Japan, March 2005
- [6] <http://www.aps.anl.gov/xfld/SoftDist/swBCDA/jca/index.html>
- [7] <http://caj.cosylab.com/>

²channel characteristic is always mapped to a certain PV field in EPICS database