THE NEW ABEANS AND COSYBEANS: CUTTING EDGE APPLICATION AND USER INTERFACE FRAMEWORK

Igor Verstovsek*, Janez Dovc, Miha Kadunc, Jernej Kamenik, Igor Kriznar, Gasper Pajor, Mark Plesko, Ales Pucelj, Gasper Tkacik. Jozef Stefan Institute and Cosylab Limited, Ljubljana, Slovenia

Abstract

The new Abeans and CosyBeans are in their third generation, developed on the basis of experience gained from several projects, such as ANKA light source, ALMA radio telescope project at ESO and NRAO and from cooperation with control groups from DESY, RIKEN and elsewhere. Abeans form an application framework and CosyBeans form a GUI framework, designed for developers of control systems of large experimental setups. These allow simple development of complex applications, providing all the benefits of the underlying Java platform, such as Java logging, high-speed graphics of Java 1.4 and easy installation and maintenance with Java Webstart. Abeans is a library that provides simple Java beans for connection with the control system. At the same time, it provides 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. By means of Adapters that act as a mediator between the underlying remote data layer and top-level GUI, CosyBeans decouple presentation of data from the actual data sources. In this manner, greater portability and consistency are achieved. Both frameworks can be used alone independently but really excel when used together, enabling a coherent look of applications that run on different protocols. We have provided plugs for CORBA, EPICS and TINE. We present some of the main Abeans and CosyBeans concepts, such as services, Abeans engine, Models, Plugs, Displayer, Adapter, CosyPanel and Plugin.

1 INTRODUCTION

The concept of Abeans as an application development framework is not new. Abeans (now called Abeans Release 2) are already used and thoroughly tested in such different environments as a synchrotron light source (ANKA, Germany), an e-p collider complex (DESY, Germany), proton cyclotron (Riken, Japan) and a radio telescope array (ESO, Germany) [1].

Experience gained from these projects has indicated the need for a new Abeans design, named Abeans Release 3 [2]. Over the last years, our group has worked extensively on this project. The result of the development are two fully functional, complementary frameworks – Abeans and CosyBeans.

In this article we will outline the main concepts and features of the two frameworks. In short, Abeans handle the modelling of complex systems and data exchange with such systems, while CosyBeans take care of data visualisation and provide the application outline.

*igor.verstovsek@cosylab.com

2 APPLICATION FRAMEWORKS

Abeans and CosyBeans are frameworks for application building. This means that an application programmer should be more productive when using both frameworks than by building applications by him/herself from scratch. We define more productive as packing more functionality into the product per unit time, achieving greater maintainability and allowing easy extensibility. In order to deliver on these promises, the frameworks must ensure the following:

Firstly, frameworks factor out shared functionality. In addition to moving pieces of code that performs the same task from multiple locations into one location, this also involves identifying functionality that *could* be performed by existing code and designing new code to benefit from the existing one.

Secondly, the nature of a framework is open-ended: it can be used in unforeseen contexts. Because it is impossible to design for this constraint, we provide simple, often light-weight default implementations, which can be replaced. This implies modular architecture that can be realized with careful design, using the strategy, component and object factories patterns.

Finally, quality assurance means that components work well and consistently together. Besides things trivial to enumerate (but harder to produce) like documentation, the framework should **force** (through design contracts, interfaces that are mandatory for implementation and by careful checking of user input) the user to adhere to certain preset standards of quality.

Without much further philosophising, we can say that the three above points bring enough benefits, if fulfilled, to warrant the use of application building frameworks.

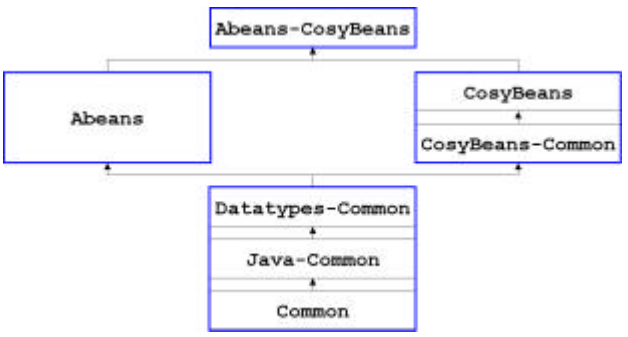

Figure 1: Schematic diagram showing the organisation of code. From bottom to top follow: Common, Abeans and CosyBeans and the integration of both.

3 PROJECT STRUCTURE

We divided all software into several projects; the organisation of is displayed on figure 1. The foundation for both application frameworks, the "Common", is displayed on the bottom of the figure and is described in more detail in the following two subsections.

3.1 Common and Java-Common

The new release introduced automatic builds for our Java projects. They are built with Ant build tool [3]. Ant is used because it is easier to manage than makefiles and because it is customized for use with Java. We have adapted Ant by writing additional Ant tasks that allow us to use one generic buildfile for all Cosylab projects, where the details for each project are stored in one place – namely SQL database – only [1]. The details include **ALL** relevant project data: developers, releases, dependencies, external jars, etc.

The common software also introduces code used by both frameworks, e.g. implementations of fast basic programmatic data structures, utility classes used for debugging, etc.

3.2 Datatypes-Common

The Datatypes library provides a consistent and extensible mechanism for producing object models of data sources. Datatypes are based on the concept of a *Property*, a logical object that groups together atomic data items which logically belong together and the data access modes for these data items. It further classifies the data items and data access modes and provides a mapping into Java language for each of these classifications. Datatypes library is referenced by both Abeans and CosyBeans.

The purpose of having an abstract property data type is the ability to use the same object, without transforming it into other objects (and thereby creating new Java object instances) at all levels of the client program, from the plug up to the visualization layer. We have decided to provide interfaces for basic types – long, double and string and for sequences of them in order to support fast, typed data transfer, and at the same time cut down on development and maintenance costs by eliminating functionally duplicate types (like float and double together). In addition, we have also provided a *PropertyGroup* – a hierarchical structure where the properties can be assembled in one of the basic forms – list, table or tree. PropertyGroup is especially handy for efficient visualisation of large numbers of control system parameters.

4 ABEANS

Given some complex software system, let us say a distributed system or for example a database, Abeans can be used to build a *model* of the complex controlled system, to build a *plug* for communication with the complex system and to organize *services* not related directly to the complex system, but to task of application building. We will look at all this concepts in more detail in the following subsections.

4.1 Models

Abeans provide the building blocks for constructing an object-oriented representation of some complex system. If Abeans are to be used for controlling physical devices, such as power supplies, vacuum pumps and so on, they can be used to create power supply objects, vacuum pump objects, etc. This includes defining the containment of such objects (objects as in object oriented programming), their lifecycle and bindings to other services offered by Abeans.

We have already implemented one fully functional model – Channel access that models the remote data as channels and is well suited to model systems that are not organised hierarchically, like for example in EPICS or TINE protocols.

In the near future we will implement a model that will allow structured data – a set of properties can be part of a larger entity, a *device*, that mirrors the logical structure of physical devices, like power supplies, etc. This model will be called BACI access model, and will be similar to the organisation now adopted in Abeans Release 2 [1].

4.2 Plugs

Usually, Abeans will communicate with some already existing software that offers access to the data. For example, the "complex system" that we keep mentioning could be a synchrotron hall full of physical devices, or an array of radio telescopes. But in addition, the "complex system" could also be a layer of hardware and software just above the physical devices, that provides data access to the devices. Usually, such layer is realized as a field bus, or an in-memory database or any other kind of structure (even for example ORACLE database) that offers to exchange the data with the rest of the world usually over Ethernet (with internet or some proprietary standard). A plug is that part of the Abeans system that is responsible for translation of requests originating in an Abeans model to a communication protocol used by the complex controlled system.

So far, we have implemented plugs for TINE protocol for DESY at Hamburg and an EPICS plug that communicates to the JCA library [4]. This plug was developed primarily for the SNS project. In addition, the same EPICS plug can also be applicable to any other EPICS-based system! In this manner we have effectively **joined Abeans and EPICS**, giving the application developer the best of both worlds.

Together with the BACI model we will implement a plug for ACS CORBA based system, and in this manner effectively finish porting of Abeans from Release 2 to Release 3.

4.3 Abeans Simulator

A very important addition to the Abeans Release 3 is the Simulator plug. A simulator is a plug that can simulate any kind of response to an Abeans request, thus

mimicking any remote system for any model. The tree of simulated objects is built either from an XML file when Abeans start (each application can have its own simulated objects), or it may even be built during run-time. Failures can be simulated as well, including timeouts, exceptions and error codes. The structure of the response can be exactly prescribed, including response times, values and all other relevant Abeans parameters. Abeans automatic tests use the simulator to ensure the integrity of data transfer through Abeans libraries.

4.4 Services

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 fulfils the requirements enumerated in the introduction, but also guarantees consistent behaviour, look and feel and functionality across all applications developed with the Abeans framework.

So far we have implemented the following services: Authentication service, configuration service, data resource service, debug service, exception handler service, report service and thread pooling service. The details are beyond the scope of this article and can be found in the Abeans documentation [1].

In addition, Abeans 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 Nameserver or ACS Manager) is done through standard JNDI (Java Naming and Directory Interface). Consequently, we can write one single directory browsing Abeans application, which will run on all Abeans plugs! Similarly, distributed archive service can access archive data in remote archive servers. Archive reader application developed for DESY uses this Abeans service. However, since the same service will be implemented for EPICS as well, we will be able to reuse the same application without modifications to access EPICS archives.

4.5 Abeans as Clients and Servers

Usually, the complex controlled system with its hardware and software layers acts as a data provider and therefore a server, and Abeans act as a data consumer, or a client. Clients are often regarded as being graphical user interface (GUI) applications. While that may be true, **Abeans themselves do not provide GUI functionality.** They merely serve as a modelling and data access library. To put it another way, Abeans are a layer that runs on the client machine, that accesses the data of the controlled system through the plug and offers it to Abeans users through the model. An Abeans user might be some visualization library, such as CosyBeans – in this case the final result of Abeans and CosyBeans together will be a graphical application. By saying that Abeans do not provide GUI functionality we do no imply that Abeans does not prepare data for visualization. Abeans have been designed to be used as Java Beans – hence the name – in visual composition tools as invisible components.

An Abeans user might be some calculation package as well: such package could get the data from Abeans, do some calculations and in response, perform certain actions back on the control system. For example, an accelerator computation package XAL of the SNS project [5] defines structures that represent parts of the accelerator. After integration with Abeans, all of communication (EPICS channel) management is handled by Abeans. This simplifies XAL implementation a great deal, since proprietary handling of channel management, JCA calls, callback registration, error handling and message dispatching, all of which were previously implemented not only by XAL but also in part by applications, are now all handled by Abeans, making XAL implementation much cleaner and job-specific.

5 COSYBEANS

5.1 Adapters and Displayers

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.

We have already ported the implementation of Displayers already used in previous release of Abeans, such as Gauger, Slider and Ledder to Release 3. It is important to stress that for all the projects we extensively use the new features offered by the latest release of Java 1.4 platform, such as assertion checking, high performance 2D graphics, anti-aliasing, etc. Besides these we have also developed some new displayers, like Wheelswitch and Date Selector.

5.1 Utility Graphics Components

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.

5.3 Application Outline

CosyBeans go even further than just providing visualization for data items. Two important concepts are introduced and fully implemented: a *launchable* and an *engine*. A launchable is a visual representation of the application. It is the application's main window. Within this basic definition certain implied functions are already hidden. The launchable does not prescribe how it is launched, it can be started as a stand-alone application, as an applet, within an internal frame of via Webstart. Implementation for this is already provided, the user just has to select a desirable launch scenario. Even more – if desired, a new application can start in the Java Virtual Machine (JVM) of a launchable that is already running. Our experience from ANKA shows that on some machines there are usually over 5 small applications running in parallel, thus consuming over 20 Mb per application, of which a great majority falls to core Java libraries, together this means over 100 Mb for all the applications together. By means of JVM sharing, these applications can now run in the same virtual machine and consume only about 25 Mb of memory in total!

An engine is the logical or processing part of the application. The engine and the launchable communicate, as a design contract, through JavaBeans events and properties. Such design enforces the separation of visualization and data access parts also on the application level.

6 INTEGRATION OF ABEANS AND COSYBEANS

From the discussions in the preceding two sections it may seem that Abeans and CosyBeans are tightly interwoven. In fact, that is not the case at all – Abeans library compiles without CosyBeans and vice versa, so there are no cross references in the libraries. Not only on syntactical level, also semantically each library can be used alone. CosyBeans engine can be a facade not only for Abeans services but for the services of some other framework, should it be written. Abeans as well can pack the data in a format suitable for display on some other visualization library, different from CosyBeans; or it could, as mentioned, be run as a server as well.

However, a true pleasure for the application developer is to use the two libraries together. Formally, the only pieces of code that reference both Abeans and CosyBeans are located in the Abeans-CosyBeans project. Herein we find specializations of CosyBeans engine for Abeans, the Abeans engine, CosyBeans Adapters for Abeans and so forth.

Abeans engine provides a facade for all Abeans services. While Abeans services are designed with maintainability and extensibility in mind, they are not particularly easy to use by themselves. They are accessed through various interfaces, they provide a lot of information that is not needed during normal application operation (as opposed to development or testing stage) and they mix regularly used functions with advanced functions. By providing the facade pattern, a developer gains a real productivity increase.

7 QUALITY ASSURANCE

Quality assurance (QA) and quality control issues are of central importance for the application framework developer. The frameworks must be well documented – besides reference and users manuals also step-by-step tutorial must be provided. We devoted a lot of effort to documentation to make our product better in terms of QA.

Besides, the code must be tested in advance because the same code will be used extensively in very different projects. Every change in the core of the framework might potentially influence some unknown application built on top of it. It is therefore of extreme importance to have a consistent set of tests of all the possible features and scenarios that are run every time something is changed in the code. This is only possible by performing automatic tests that are run every time the code is built. We use a testing framework for Java called JUnit [6].

Of course, undetected bugs can always be found by the CS operator. In addition to usual support, we are also planning to develop a plugin for CosyBeans that will, in case an unexpected exception occurs, write an automated mail to Cosylab support with exception stack trace and other relevant data, together with optional remarks entered by the user that has found a bug. We use an effective bug reporting and managing tool, the *Request Tracker* [7]. Thus, the more people will use Abeans and CosyBeans, the better they will get.

8 FRAMEWORKS AT WORK

We tried to show that our application frameworks are very functional, provide all the features one expects from them and much more – to sum up, that our frameworks are very good. Nevertheless, for the application frameworks to be a true success, they must be widely used.

Figure 2 and the table below show the amount of work invested into Abeans and CosyBeans frameworks and into the TINE project for DESY that is built on top of them.

Product	**Lines of Code**	**Estimated Work**	**Estimated Cost**
Common	7,343	1.58 my	$ 213,000
Abeans	26,715	6.30 my	$ 851,000
CosyBeans	44,931	10.54 my	$ 1,423,000
Integration	4,214	0.91 my	$ 122,000
Together	**83,203**	**20.76 my**	**$ 2,804,000**
TINE	14,120	3.22 my	$ 435,000

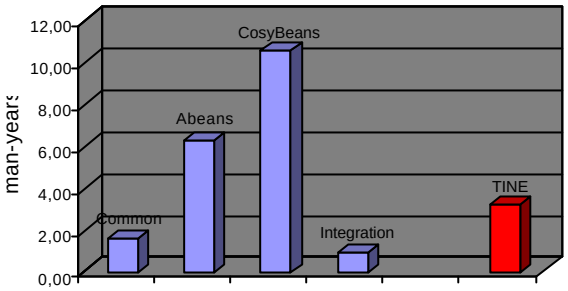

Figure 2: Estimation of work invested into the development of the frameworks. Development of the TINE project is shown as a comparison.

SLOC stands for Physical Source Lines of Code, from which also the amount of work was estimated. The statistic analysis program that was used was SLOCcount [8]. It uses a model for the estimation of work time and costs. This model is not linear, meaning that the work calculated for completion of all frameworks is larger than the sum of work for the development of individual parts. This compensates for the overhead when developing more projects at once. Average salary was taken to be about $ 56,000 / year with an overhead factor of 2.40. It must be emphasised that the cost estimation is very crude, we (unfortunately for us) did not get paid even a fraction of the calculated price. Nevertheless, the following conclusions can be drawn from the data:

- The amount of common code, described in section 3, is relatively small. The largest amount of work falls to Abeans and CosyBeans. Integration of the two frameworks is very lightweight, speaking in favour of the overall design.
- Total investment of time into the frameworks is substantially larger than the time usually devoted to the development of the corresponding part of a typical control system project. This implies that the development of the framework is not justified for a single project. Hence, the development of our frameworks is a significant contribution to the CS community.
- Project TINE for DESY, which contains Abeans plug for TINE protocol, two complex applications Instant Client and Archive Reader and a sophisticated ACOP chart by DESY. The total amount of this code is very small compared to the complete framework, which speaks in favour of Abeans and CosyBeans – it is easy to write plugs and applications for a given control system.

It must be noted that actual development time was about 8 man-years, which is less than the estimated 21 man-years. This indicates that our team is very efficient – the reason lies in our long-time experience with the development of application building frameworks.

If we take the general rule [2] that the effort to build a framework must be offset by at least three times the amount of work invested in the application writing approach to pay off, the Release 3 has still a way to go before we will be able to say that the development of the frameworks was justified.

However, the things are looking very bright for the future of Abeans and CosyBeans – we are currently implementing a plug for the CS of the SNS project based on EPICS, we will port Release 3 to an ACS CORBA based CS that we are currently developing in collaboration with ESO [9] and we also plan to install Release 3 in ANKA.

9 CONCLUSION

In the preceding sections we have demonstrated the advantages of the third generation of our application development frameworks. A colleague from SNS said that Abeans and CosyBeans frameworks solve problems that they did not even know that existed, but were sure they would found them at later stages of development had they not chosen to use Abeans and CosyBeans. We believe that Abeans and CosyBeans are now mature and ready to be used extensively in any professional client application.

All in all, it is a much more fulfilling undertaking to write an excellent application framework than it is to write tons of similar applications, solving the same problems time and time again.

10 ACKNOWLEDGEMENTS

We would like to express our sincere thanks to colleagues at DESY and SNS for their hospitality, useful discussions and continuous support with the development of the frameworks. We would also like to thank the members of the ESO team for valuable suggestions and stimulations.

11 REFERENCE

[1] Cosylab homepage, http://www.cosylab.com and KGB project homepage, http://kgb.ijs.si/KGB

[2] Gasper Tkacik, et al., “How to Build Professional Control System Applications“, PCaPAC 2000, Hamburg, October 2000

[3] Jakarta Ant homepage, http://jakarta.apache.org/ant/

[4] Ales Pucelj, et al., “Design Considerations for Integration of Wide Range of Control System Communication Protocols in Cross-Platform Framework on the Example of EPICS and TINE”, PCaPAC 2002, Rome, October 2002

[5] SNS project homepage, http://www.sns.gov

[6] JUnit project homepage, http://www.junit.org

[7] Grega Milcinski, et al., “Developing a Control System from a Divan Bed”, PCaPAC 2002, Rome, October 2002

[8] SLOCCount project homepage, http://www.dwheeler.com/sloccount/

[9] Mark Plesko, et al., “ACS, The Advanced Control System“, PCaPAC 2002, Rome, October 2002