DISTRIBUTED COMPONENTS IN CONTROLS

M. Plesko, B. Jeram, I. Kriznar, B. Lesjak, T. Milharcic, R. Sabjan, G. Tkacik,
I. Verstovsek, K. Zagar; J. Stefan Institute, Ljubljana, Slovenia, http://kgb.ijs.si

Abstract

We have built a control system framework that uses and extends modern component-based, distributed computing and object-oriented concepts. The basic entities of the system are accelerator devices that are represented as CORBA objects. Each device is further composed of property objects that correspond to what is called channels in EPICS. Properties support get/set commands, event-driven monitors and alarms, asynchronous and synchronous communication, logging, access to the database, etc. As all devices and their properties have common functionality, the servers for those CORBA objects are generated with a wizard-like program generator. Another generator creates one Java Bean component for each CORBA object. The client applications, written in Java, are then composed of those "accelerator components" in any commercial development tool, often using visual and graphical programming. The Beans are essentially components wrapped around CORBA, because standard CORBA components do not exist as yet. However, they are more powerful as they contain a pluggable interface that connects to any communication framework apart from CORBA, such as CDEV or even a generic simulation of devices and properties. The plugs are determined and switched at run-time. Generic applications that cover all basic needs including machine physics applications have been made using CORBA dynamic invocation and Java reflection. The functionality of the system is rounded up by a series of tools based on a remote management framework, which is composed of security/authorization mechanisms, object and process management, configuration database versioning and rollback, control event logging, and similar.

1 INTRODUCTION

The presented distributed component framework forms the core of the control system for the light source ANKA at the Forschungszentrum Karlsruhe in Germany. The 2.5 GeV storage ring of ANKA is just being commissioned and should be operational by spring 2000. The ANKA control system[1], which is completely outsourced to the J. Stefan Institute, builds on the three-tier standard model architecture. Modern products based on standards in distributed objects and networking are applied in addition to low-cost hardware including PCs. The LonWorks field bus network with intelligent nodes and standard I/O modules connect the individual devices directly to PCs. The communication to these “device server” PCs is done through CORBA[2]. Other PCs on the net run Java clients that wrap CORBA objects into Java Beans[3], which are then connected with commercial data-manipulation and visualisation Beans using visual tools or programmatically. The CORBA objects and Java Beans are generic models of controlled data that can be used at any other control system.

2 OBJECT ORIENTED COMPONENTS

2.1 Concepts

A crude definition of software component is “a reusable piece of code”. However, such a product developed for the in-house use has little chance of being adapted for general use and re-use even if it is free and efficient, unless it fulfils the following conditions:

- it is easy to use, which includes being easy to learn
- it employs standard programming practices and adheres to well-defined and established frameworks
- it is flexible in the sense of being easily ported, adapted or extended

Some of these requirements are satisfied automatically by the object-oriented programming (OOP) practice. Ease of use, however, necessitates another step, that is the use of component technology (OOP that uses existing component frameworks and sticks to certain design guidelines / patterns).

Our components - Accelerator Beans or Abeans[3] for short - for rapid application development of accelerator control client software meet these requirements by:

- using Java Beans technology, thus enabling the user to build the panels visually, while the development environment generates the code
- enforcing strict separation between components that have no visual representation but represent the communication, error handling, synchronisation and similar frameworks; and components that offer strictly visual capabilities but are not aware of the fact that remote objects are being controlled through them
- separating the communication system dependent functionality into separate modules, called plugs, which can be dynamically loaded: porting Abeans consequently requires only the plug (interface to the underlying control system) to be written, while all

upper layers (frameworks, visual components and applications) remain totally unaffected.

2.2 Component Hierarchy

When writing a client application the programmer is most often confronted with numerous problems from the areas of error handling, timeout handling, logging, communication system details, resource initialisation and destruction and the like. In terms of lines of code it can easily happen that such functionality easily compares with the core application functionality. The same reasoning applies to GUI programming. A divide-and-conquer solution that ANKA client software embraces is the use of the following approaches:

- Control system is comprised of server layers and client layers; the latter consist of framework (invisible) Abeans layer (which in turn consists of server-dependent or pluggable layer and independent layer), and application (visible) layer
- There can be many types of controlled devices and therefore many different interfaces (one device - one class approach), however, each device is constructed of fundamental building blocks - Properties, and there is only a small number of these (RWDoubleProperty represents a physical quantity of double type that can be either changed or retrieved, for instance a current in a power supply; there are currently 3 such types in ANKA CS: RWDoubleProperty, RODoubleProperty, ROPatternProperty, representing a retrievable bit-pattern)
- The core visual components (such as gauges, status displays, trend charts) should be completely independent of the actual control system, so that they can be used by others
- Special adapters have intimate knowledge about the Property and about the visual component, allowing a generic visual component to be "adapted" to the Property. This means that the Property will be an invisible framework class and the generic visual component will, with the help of an adapter, become its view (to use the Model-Controller-View terminology).

In this way, the learning curve for a new programmer is not very steep and demanding. For instance, ANKA control system uses CORBA for communication, but programmers writing the visual components or "views" are completely unaware of CORBA, since the invisible Abeans layer hides it. All interactions with the data sources that the view needs to display its data is performed through simple and well-defined Java Beans interfaces.

The pluggable layer offers the possibility of accessing different remote systems and by its nature of adapting the lower-level remote system interface to the Abean interface enforces uniform and consistent behaviour across a range of possible remote systems.

3 CORBA OBJECTS

We believe that the most human-oriented way is to represent each accelerator device with a respective object. CORBA allows very fine-grained definition of objects and their properties. With a control system we control accelerator devices, such as power supplies, beam position monitors, etc. So it is natural that we represent those devices in the computer with objects. In the network, they become CORBA objects.

The example analysis of a power supply in Table 1 shows how a device object looks and what kind of properties it has.

Table 1: properties and commands of a power supply:

Property	**Type**	**Access**
ADC:	double	Read only
DAC:	double	Read/write
Status	bits	Read only
Command	**Input type**	**Return value**
on():	none	void
off():	none	void

We see that properties have different types and also differ in whether they are read-only or also can be written to. But there is more to properties. A property, such as power supply current, has properties of its own: a minimum, a maximum, a description, units, etc. To avoid confusion, the properties of properties are called characteristics.

In most cases, all properties have the same type of characteristics. There is a difference between read-only and read-write properties, as the former have no command to write. Another difference is between properties of type double and type bits (actually an unsigned integer containing a bit pattern), as the latter have no minimum or maximum. But all these differences can be systematized into a matrix of few classes[2].

4 THE IMPLEMENTATION

Although the control system is designed to support all primitive data types for controlled values, only double and unsigned integer did suffice. In turn, the number of necessary GUI components is small. Not only were the types of values limited, also their properties were defined and fixed in advance. This means that a controlled value is either read-write or read-only and that it has a name, a minimum, a maximum, etc.

4.1 Generators

The list of properties was made large and configurable enough to cover .all cases, yet the sole fact that it is fixed

in advanced simplified the design of clients and servers and databases.

Also the device servers and the API are then device specific, however due to the pre-defined value types and structures, all those components can be built easily and fast. In fact, a wizard has been written which generated device servers based on device interface definitions. The Java API libraries, JavaBeans, need less than one page of extra code per device. So in total, the concept proved to be advantageous.

4.2 System Management

The functionality of the implemented object management would shortly be described as:

A: view current state of control system

- state of Device Servers
- name of computer, where Device Server is running
- shows exported and exportable Devices of each Device Server
- state of Devices

B: management of Device Servers

- download component database to local disc where Device Server is located
- change of component database location
- shutdown, restart

C: management of individual Devices

- destroy, export of Device with DeviceServer
- activate, deactivate
- show and manage all clients connected to Device

4.3 The Component Database

The component database stores hierarchically organized data. Data may be in form of simple type, such as String, Double, Integer, or organized in arrays or objects. Data are stored in text files (*.txt) that are placed in structure of folders. The path to each piece of data consists of directory path, name of file and it's declaration in file. The files may be edited manually with text editor, such as Notepad. The data stored are physical characteristics of accelerator devices and properties needed to communicate with devices. This data is static, it doesn't change much, except when new devices are added or old removed. Therefore the component database is also called staticDB. The database is located on local computer, where device servers are running.

A device server reads data from StaticDB in form of attributes. An attribute consists of path name and value. The value of an attribute can have data type, simple or complex as Array or Object. The path name is description of location in StaticDB architecture where value is stored.

Expressions and variables can be used for Integer and Double type. Supported operations are: +, -, *, /, ^, sin(), cos(), sqrt() and exp(). Variables are all names of values defined in the same file. In language of paths this means that as a variable in an expression any node name can be used that has the same parent node as the evaluated node.

Templates are used when a group of attributes has same data value or to provide them a default value. In such case is wise to store this value in one place, so only one rending is necessary when value should be changed. Templates also ease StaticDB managing and building. When StaticDB can't find an attribute value in the given path, it will look into template locations that are defined inside the attribute paths. There can be more then one template location.

For version control of the database Microsoft Visual SourceSafe is used. The central version of StaticDB is stored as a SourceSafe database on one computer in the local Microsoft network. The recommended tool for editing files is TextViewer (Textview.exe) which is included in NT Resource Kit. On the left side of the program window is a visible structure of folders and files in StaticDB, on the right is the visible contents of selected file. For Reading and comparing StaticDB two tools are provided: **DBReader** reads single attributes from StaticDB; **DBCompare** analyzes the whole StaticDB and prints out selected attributes, all attributes or all unreadable attributes. DBCompare also compares one StaticDB to another.

CONCLUSIONS

Experiences with the running system for the ANKA light source show that the concept lives to its promises. A successful port of the Java Beans to a test case at the Swiss Light Source, where the calls to CORBA were replaced with calls to CDEV, is a proof that the strict object oriented model of devices can be used to effectively hide other types of communication channels.

The Abeans present the final user with a view of the accelerator as a collection of accelerator beans, thereby reducing the problem of writing applications for the control system to the problem of familiarizing oneself with the ways to use java beans and development tools.

All interfaces that Abeans expose are type-safe (with no 'single function taking control string' methods) and enable one to construct a simple yet powerful graphic application within minutes.

REFERENCES

[1] M. Plesko et al, A Control System Based on Web, Java, CORBA and Fieldbus Technologies, PCaPAC99 workshop, Tsukuba, January 1999.

[2] M. Plesko, Implementing Distributed Controlled Objects with CORBA, PCaPAC99 workshop, Tsukuba, January 1999.

[3] G. Tkacik, Java Beans of Accelerator Devices for Rapid Application Development, PCaPAC99 workshop, Tsukuba, January 1999.