THE CORBA IDL INTERFACE FOR ACCELERATOR CONTROL

M. Plesko
J. Stefan Institute, Ljubljana, Slovenia, e-mail: mark.plesko@ijs.si

Abstract

It is a long practice that hardware designs and solutions are shared among accelerators, because well defined standardized interfaces exist. In order to be able to share software among different accelerator control systems, a standard library or application programmer's interface should be adopted. A proposal for such a standard description of accelerator devices is presented in this paper. It is a language independent collection of interfaces based on network distributed objects using the CORBA standard. All common accelerator components such as power supplies, vacuum, RF, position and current monitors are defined by means of functions and parameters. The interface does not replace or compete with any of the existing accelerator control systems (EPICS, CDEV, TACO, DOOCS). On the contrary - great care has been taken to be as compatible as possible to those systems so that all could use the CORBA interface.

1. INTRODUCTION

The Accelerator CORBA Interface (ACI) [1] defines controlled devices (e.g. power supply, current monitor, vacuum pump, etc.) as network objects that are remotely accessible from any computer through the established client-server paradigm. The underlying communication mechanism is based on the Common Object Request Broker Architecture (CORBA), the state-of-the-art standard for remote objects[2]. The devices are described according to CORBA with the Interface Definition Language (IDL), which presents a language-independent way of defining object interfaces.

The ACI is meant to be a standardized interface so that applications and pieces of control systems can be hooked to it from either side. The ACI does not replace existing control system architectures and frameworks.

The ACI attempts to build a model of devices that are commonly used in all accelerators. It is the largest common denominator that can be found among different types of accelerator facilities. As such it should enable portability of control software and ultimately reduce the dispersed efforts at various laboratories where the same software is written all over and over.

The ACI is but a definition of device interfaces with the use of IDL, not a definition of an API. An API with more powerful or sophisticated features can be built atop of the IDL interfaces, or even replace the CORBA protocol with a proprietary scheme. However, the IDL interfaces have been defined such that it is possible to perform all necessary control actions just by direct CORBA connections to the ORB that exports the IDL device interfaces. The idea behind this approach is that it is not necessary for the client to load any special API library – any CORBA ORB can find devices on the net and communicate with them.

2. DESIGN GOALS

The design goals of ACI are:

- Rely on pure CORBA only: don't be language or system specific; don't assume extra functionality in an API library.
- Enforce strong type checking wherever possible. Illegal commands should be discovered already during compile time. Run-time parsing of commands through constructs like send("command") must be avoided. Generic applications can use the introspection capabilities of CORBA (e.g. Interface Repository, Dynamic Interface Invocation, etc.) instead.
- Exploit the object paradigm: the object itself is responsible to provide all data that is relevant to it. Avoid therefore direct access to database servers; leave this to the implementation.
- Don't try to define a generic interface for any possible control system. Specialize on the definition of accelerator objects with the functionality that is common to all accelerators.
- Define object interfaces; don't prescribe their implementation and don't provide client-side functionality. The ACI is merely a hook to the underlying control system.
- Don't allow the client to manipulate control system behavior. Assume rather that reasonable default values are provided by the system managers through control system configuration tools.
- Use well-proven concepts from existing accelerator control systems.
- Base data transfer on asynchronous calls assuming that all client and server host operating systems are multithread capable as is necessary for GUI-based applications. Keep synchronous calls just for compatibility with legacy systems.
- Encourage site-specific additions through interface inheritance instead of providing generic bypasses to strong type checking. However, all interfaces that are defined in the ACI must be implemented at a given site, because client applications from other sources rely on them. Clients written on site can still use the added functionality without penalty.
- A technical issue inspired by Java: pass all parameters to methods by value; in IDL this means that all parameters are declared as "in". In order to save space, the "in" keyword is omitted in all definitions in this text.

Great care has been taken to be as close as possible to

existing control system frameworks like EPICS, CDEV, TACO, DOOCS, etc. Many concepts were actually taken directly from one or several of those frameworks. A few examples:

CDEV	devices are objects that are a collection of properties
EPICS	each property has a set of standard characteristics
CDEV	groups are used to merge several commands into one message
TACO	a server manages the interface for one type of devices
DOOCS	the object is keeping short term history data
all	synchronous and asynchronous get/set calls are supported

3. CONCEPTS OF ACI

This section defines all objects and concepts that are present in the ACI interface definitions (see Ref. [1] for a complete description of the interfaces). The naming has been adopted according to the convention accepted at the SOSH98 software sharing workshop[3].

3.1. Devices

A *device* is a CORBA object that corresponds to the model of a physical device, e.g. power supply, vacuum pump, current monitor, etc. A *device server* is a CORBA ORB that implements one specific device interface and exports several devices that are distinguished only by their name. Although one device server always exports only devices with the same interface, the inverse is not true. It is possible that several different device servers export the same interface. A typical example is an accelerator complex with an injector and a storage ring. The device server for the injector exports the power supplies of the injector, while the device server exports the power supplies of the storage ring. Both types of power supplies have the same IDL

The device is the basic entity of the ACI, because it is the most natural concept for modeling physical entities in an accelerator. *Commands* that are executed on a device, like on, off or reset are correspond to methods of the device. Each device has a number of *device properties* that are controlled, e.g. electric current, status, position, etc.

Device properties, which are also defined as objects in the ACI, are referred to as IDL attributes of the device. Properties are distinguished by type (pattern = unsigned integer, double, etc.) and by being read-only (RO) or read-write (RW) objects. Each such "property object" has specific *characteristics*, e.g. the value, the minimum, its description, units, etc. The methods of a property allow to retrieve or modify these characteristics: get(), set(), minVal(), etc.

Finally, a device has *resources*, which are implementation dependent static key-value pairs, like "position"-"sector 3", "interface"-"analog 16 bits", etc. Resources are not used by the ACI and by generic applications. However, they are provided as a generic way to retrieve information that is not covered by the ACI from a database.

An example of a definition of a device is:

```
interface PowerSupply : Device {
        // properties
        readonly attribute RWdouble current;
        readonly attribute ROdouble readback;
        readonly attribute ROpattern status;
        // commands
        void on(CBvoid);
        void off(CBvoid);
        void reset(CBvoid);
}
```

The interface of the power supply extends the generic object Device (which has a name, metohds to acces resources and other general properties and methods) and contains 3 properties: current, readback and status. The power supply can execute 3 commands: on, off and reset.

3.2. Data Types

Each value read from the control system and each completion of a command has an associated *error type*, *error code* and a *time-stamp*. ACI defines a basic set of error types and codes for errors and alarms. Given that CDEV, EPICS, TACO and DOOCS have their own error codes, it might be possible to either translate common errors to the code list of ACI, or to just pass the error code and indicate by type which system the code comes from. But that would make error handling of clients more complicated. The time should be ideally represented by the CORBA time service through the UTO interface (which is **not** the POSIX time). As the time service is not yet part of all ORBs, we use the CDEV definiton of time which is a double of seconds elapsed since January 1st, 1970.

Several values of the same type are stored as a *sequence*: longSeq, doubleSeq, etc. Such sequences are already provided by the IDL syntax. as

```
typedef longSeq sequence long ;
```

Sequences are used when multiple devices are controlled with one method call or when a history of values of one property is requested. The use of sequences for individual values is possible but strongly discouraged, as properties are supposed to be simple objects related to one I/O channel.

3.3. Callbacks, Monitors and Alarms

Most of the device commands and property methods are executed asynchronously by the remote object. The results of the operations are communicated to the client by means of a *callback*. A callback is an object interface that must be implemented by the client, so that it can be invoked by the remote object. During this process, the remote object functions as a client and the client performs as a server.

A client will often need to get the value of a property on a regular basis, either at given time intervals or whenever the value changes. A regular callback with the updated value is invoked by means of a *monitor*. The client creates a remote monitor on the server with a single call, where a reference to a callback is passed as a parameter. Then the monitor on the server invokes the callback whenever the requested conditions are met. An important type of monitors are *alarms*: A client registers a callback which is triggered every time an alarm condition appears, changes, or disappears.

3.4. Groups and Structures

Devices are logically arranged into *groups*. Groups are container objects that contain zero or more devices from the same device server, i.e. have the same interface definition. A group can not span several device servers. A device can be a member of more than one group. E.g. the power supply of horizontal corrector #3 is member of the following groups: powerSuppliesGroup, and corrHorGroup.

The concept of groups is inspired by the CDEV group mechanism, although it is not quite the same. CDEV uses groups to group a series of arbitrary commands that should be transferred to servers in one network message. This very generic feature can not be defined through a remote interface, which the ACI is. However, the most common usage of groups is when the **same** device property is read or written to a group of equal devices or the **same** command is execute on them. That can be easily defined through an interface like ACI.

For tasks that span several device servers, groups can be arranged into *structures*. A structure is a collection of groups from different device servers. A typical structure is a "Storage Ring" or a "Transfer Line". Structures are handled by a dedicated CORBA server, which communicates with the device servers, where the groups are created. Alternatively, structures can be defined also on the client within a dedicated API.

Groups and structures are used for two purposes:

1. Devices of the same or different type are pre-arranged within the IDL into groups and structures, respectively, such that generic applications know which devices to use for specific tasks, e.g. orbit correction, synoptic display, save/restore machine state, etc.
2. Devices of the same type (same IDL interface) are arranged by a client into a group to perform the same action on all members of a group with a single method invocation, e.g. set the current of all power supplies.

4. THE IMPLEMENTATION

The concepts of ACI have been checked on a "real world case" at the ANKA control system [4], which currently implements all but groups and structures. Besides conventional use of the ACI, the CORBA dynamic invocation interface (DII) was used to write a generic client. Another set of clients were built graphically in a visual builder with JavaBeans that expose ACI such that all details of CORBA are hidden. The complete system proves to be very efficient and conceptually clean.

5. CONCLUSIONS

The present article models accelerator devices as distributed objects using the *strong typing* paradigm at all levels. This approach provides the application programmer with powerful compile time checking, thus eliminating many errors and bugs that would appear at run-time and would be difficult to track down.

Another advantage of the present concept is that there is no need for separate API manuals and device manuals, because the CORBA interface is both the API and the list of all legal commands for a device.

The ACI is completely independent of any concrete accelerator implementation. Thus it appears that accelarator devices defined with ACI or some similar framework based on IDL and CORBA are the right basis for sharing high level accelerator applications.

6. ACKNOWLEDGMENTS

The author thanks Bogdan Jeram for many fruitful discussions and for implementing the presented ideas into a real control system.

7. REFERENCES

[1] For a complete description of ACI see http://kgb.ijs.si/Clanki/ACI_draft_3.html
[2] http://www.omg.org/
[3] http://www.sls.psi.ch/SOSH98/
[4] B. Jeram et al., The Control System for the Accelerator of ANKA, this conference.