A GENERIC SIMULATOR OF CONTROL SYSTEMS FOR APPLICATION DEVELOPMENT AND TESTING

Dragan Vitas[*], Mark Plesko, Gasper Tkacik, Ales Pucelj and Igor Kriznar

Jozef Stefan Institute and Cosylab Limited , Slovenia.

Abstract

The modern approach in software development for control systems requires RAD tools like the CosyBeans/Abeans framework. These tools enable the developer to quickly produce an application with well-tested functionality. This shifts the developer's focus to device integration, which is always case specific and needs good testing. When the new application is deployed on a real system for the first time, everybody expects that it will run smoothly and without any strange behavior. In practice this is not often the case.

Thorough tests are still necessary to verify that the application behaves according to specifications, in particular the graphical user interface. This can be done only in a realistic environment. Unfortunately, it is a common case that developers are not allowed to interfere with a running system. We have therefore developed a generic simulator that tries to simulate as realistically as possible a running system. Even erroneous behavior or situations that are not likely to ever happen in a real system can easily be produced with a simulator to test how the application will respond to them. The application programming interface of the simulator allows adaptation to different control systems and the writing of value generators, small routines that simulate a particular behavior. In this paper, we present the architecture of the simulator and different areas where we have successfully used it. Using the simulator, our developers have polished their masterpiece, improved its performance and got rid of bugs before anyone even noticed that they were there.

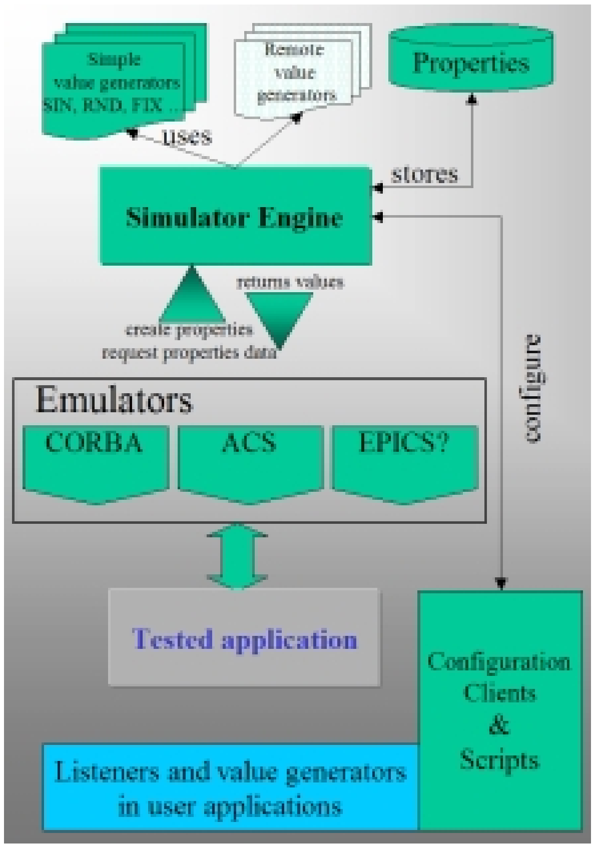
Simple value generators SIN, RND, FIX ...
Remote value generators
Properties
uses
stores
Simulator Engine
returns values
create properties request properties data
configure
Emulators
CORBA
ACS
EPICS?
Tested application
Configuration Clients & Scripts
Listeners and value generators in user applications

Figure 1: Simulator overview.

1 DESIGN PRINCIPLES

To facilitate application development and testing in our company [1] we developed a software program named Simulator. The Simulator follows few design principles:

- It must not be specific to any particular control system software.
- Its design must be flexible enough to allow the tester to simulate almost any kind of response to a given request.
- The design must allow easy implementation of new specifics.
- Applications have to be tested without being modified.
- The user must be able to write extensions in a language and platform of his choice.

The logical schema is shown in figure 1.

2 SIMULATOR ENGINE

The core functionality of the Simulator is independent of any specific control system and will be referred to as the Engine in this document. The Engine functionality is exposed as a CORBA [2] object. This is chosen for the following reasons:

- The interface is defined in a standard way - IDL file.
- Multi-language, multi-platform clients can be created.
- Advantages of the CORBA infrastructure can be exploited.

The Engine is essentially storage of name/value pairs. The name/value pairs linked with their data are named properties. The property value can be submitted to or requested from the Engine. When the property value is requested, the Engine will use a routine in order to

[*]Dragan.Vitas@cosylab.com

produce a value. These routines are named value generators. The Engine enables registration of a function that will be called when the property value is changed. These functions are named listeners. The Engine is implemented with performance and robustness in mind and because of this it is implemented as a multi-platform multi-threaded CORBA server in C++ using the ACE/TAO ORB [3]. Its main duty is to handle a large number of properties, produce their values and notify subscribed listeners. In addition it handles other activities such as creating and destroying properties, configuring their behavior, searching for properties, subscribing listeners and generators and producing listings about its configuration and state. Its persistence can be stored in a set of XML data. The data can reside locally or they can be fetched from remote XML storage using CDB (Configuration DataBase) [4].

2.1 Properties

The properties in the Engine have associated data, which are used to handle property configuration and its behavior. The data are property type, value range, current generator and subscribed listeners. The properties in the Engine are typed. This means that the Engine cares if the property is double, long or string. A property can also be of CORBA Any type. Property value can in that case be arbitrary and the Engine treats it as 'other' type. Such types are treated the same as basic types except that their values cannot be generated but are returned by the Engine as they are. This is because we want to have a simple interface for generators with a few, but enough number of types. Range is specified as an interval (from-to) in which property values are allowed, the generator is a reference to the currently chosen generator while the listeners are recorded as a simple list of objects.

2.2 Value generators

The value generator is a small piece of code that is responsible for producing property values. The Engine has a few predefined value generators:

- Sinus: Values are varied according to a sine curve
- Random: A random value is returned in a property’s range
- Fixed: The value is always the same fixed value
- Set: The value returned is the same as the value that was set for this property from the client application
- Increment: The value is incremented each time it is requested

These generators are fixed and they are implemented and instantiated by the Engine itself. Beside these generators there is one special type of generator – Remote. The remote generator is a CORBA object. Its interface is simple and straightforward to implement. Generally one virtual function is to be implemented. The remote generator is used when the user needs special handling of the property value. Typically, the Engine runs on a server while the remote generator runs on the user machine. The Engine will ensure that the remote generator will be called each time the get_value() request arrives to the Engine for the so configured property. In that case the user code is responsible for the value the property will have and how quickly it will be produced.

2.3 Listeners

The listener is a registered callback that is called each time the value of the property changes. The listener is a CORBA object, too. The listeners are typically used by the Engine clients to display current value of the property. The listener is useful if we would like to implement a value generator for a property, which depends on other properties. The listeners can be chained on a single property. The notification activity does not disturb other activities of the Engine. Even problems notifying one listener do not bother other listeners. If the Engine is unable to notify a listener after a few tries, it will automatically unsubscribe it.

3 CONFIGURATION CLIENT

The configuration client is a generic tool for basic Engine functions. It is a CORBA client for the Engine server. Through the configuration client the user can browse properties that currently reside in the Engine. The user can filter the list and configure the property. For example, he can change value range or generator of the property. We have provided configuration clients in JAVA and C++ with GUI, and as a command prompt application. This spectrum proves that the Engine implementation as a CORBA server is a good choice, since writing a client is a matter of a few function implementations, because all tricky parts for connection, network communication and parameters marshalling is handled by the CORBA. In addition there is a special possibility of communication with the Engine - CorbaScript [5]. CorbaScript is an interpreted object-oriented scripting language dedicated to CORBA environments and therefore a perfect match for the simple value generators or for the listeners. Since it is a scripting language there is no need for the user to compile and assemble the executable, and changes are in effect immediately after they are saved. The next step of Simulator development could be the integration of CorbaScript with the Engine and its export as box to the clients. In such a way users will have everything in a remote black box without any need for local software in order to use and exploit the Simulator.

4 EMULATORS

The Abeans [6] [7] framework is applicable to any control system thanks to its plugable model. Because of that, our developers can produce GUI applications that run everywhere and therefore the Simulator must follow that philosophy too. Emulators in the Simulator are software parts that come between the user application and the Engine. The emulator handles all specifics of its clients and represents a view of a control system. On the other side it utilizes the Engine.

4.1 CORBA emulator

The CORBA emulator is an object factory that creates CORBA objects, which simulate operations just by using an interface definition. Its input is an IDL file from which it creates objects capable of executing operations specified in the given interface. It relies on an Interface Repository [2] as the store for interface definitions. The CORBA emulator is itself a CORBA object with a very simple interface. Actually it has only one function:

```
interface CORBAEmulator
{
        DeviceServer   createServer(   in   string
idIDL, in string servName );
};
```

The first parameter is an IDL identifier. For example "IDL:ACS/PS/PowerSupply:1.0". It identifies the interface, which the created CORBA object should represent. The Parameter servName is used to create servers name so the client can find and connect to that object. Created name looks like 'server:port/servName' and it is described in the Interoperable Naming Service specifications [2]. The emulator first tries to locate the given IDL in the interface repository. If it succeeds it constructs an interface description. Based on that description it creates properties in the Engine in such a way that all encountered in-out, out and return values are created. For example if the servName is 'PW_SUPP_HU' and the following method description

```
long get(out double ampVal);
```

is found in the interface PowerSupply, the emulator will create

```
PowerSupply.PW_SUPP_HU.ampVal DOUBLE
PowerSupply.PW_SUPP_HU.get.returnValue LONG
```

properties in the Engine.

After that the emulator creates a DeviceServer object that is a DSI (Dynamic Skeleton Invocation) [2] server. Thanks to DSI technique, the created object can simulate any type of function with any number and type of parameters. When the DeviceServer gets a request, it scans for input parameters and puts them in the Engine, fetches return values and all output parameters from the Engine and returns the assembled response to the requestor.

4.2 ACS emulator

The basic components in the ACS [8] system are known as Distributed Objects (DOs). DOs are CORBA objects and as such they can be simulated using the CORBA emulator, but in such a way only client applications can be tested but not the ACS core itself. With the ACS emulator all parts of the ACS system can be tested including core components from BACI [9] and MACI [10]. The DOs have properties, which use the DevIO interface for accessing their values. This interface is the ideal place for the ACS emulator and the ACS emulator is actually a subclass of the DevIO interface that instead of using hardware devices, uses the Engine to store and fetch values. This emulator is in the form of a shared library that comes instead of one original ACS shared library. To run the ACS in simulation mode all we have to do is ensure that the emulator library will be located in the search path before the original one.

5 CONCLUSION

We found the Simulator very useful for evaluation of applications performance in critical network traffic situations. We use it to identify problems and critical factors as early as possible in the application development process. Since the interaction with the Simulator can be slow or fast, some application code that runs unacceptably slow can be spotted easily. In a real system you cannot tell for sure whether the problem lies in network traffic, device drivers, hardware or control system but with the Simulator you can systematically eliminate possibilities one by one and for sure spot the part of the application that is problematic. Using the Simulator we can predict application scalability and performance for a given environment and system complexity. We use it to test exceptional conditions and operations limited or restricted on a real system because of security or some other reasons.

Thanks to the Simulator our developers can do all their exhaustive tests in an environment similar to a real situation and because of that our clients get their working applications free of bugs with the first deployment and that makes them happy. Developers on the other side can work from a place of their choice, such as from their sofas [11] and that makes them happy too.

6 REFERENCES

[1] Cosylab homepage, http://www.cosylab.com and KGB project homepage, http://kgb.ijs.si/KGB
[2] CORBA (http://www.omg.org)
[3] ACE/TAO (http://www.cs.wustl.edu/~schmidt)
[4] CDB – ACS Configuration DataBase (http://kgb.ijs.si/KGB/Alma/Docs/CDB.pdf)
[5] CorbaScript (http://corbaweb.lifl.fr/CorbaScript)
[6] Abeans (http://kgb.ijs.si/KGB/Alma/Docs/Abeans_White_Paper.pdf)
[7] Igor Verstovsek, et al., " The new Abeans and Cosybeans: Cutting edge application and user interface framework ", PCaPAC 2002, Rome, October 2002
[8] Mark Plesko, et al., "ACS, The Advanced Control System", PCaPAC 2002, Rome, October 2002
[9] BACI - ACS Basic Control Interface Specification (http://kgb.ijs.si/KGB/Alma/Specs/ACS_Basic_Control_Interface_Specification.pdf)
[10] MACI - Management and Access Control Interface Specification (http://kgb.ijs.si/KGB/Alma/Specs/Management_and_Access_Control_Interface_Specification.pdf)
[11] Grega Milcinski, et al., "Developing a Control System from a Divan Bed", PCaPAC 2002, Rome, October 2002