BAE SYSTEMS
INSPIRED WORK

GimmeGimmeGimmePP

Introduction

This document describes an ECOA® client-server example, named "*GimmeGimmeGimmePP*". It is the same as "*GimmeGimmeGimme*" (ref.[4]) but the ECOA ASCs are implemented in C++, not C.

The client-server model (ref.[2]) is one of the most basic data, task, or workload, distribution mechanisms in computing. Clients and servers may be distributed across a network, or they may reside on the same computing system. Service oriented concepts, which form a basis behind the ECOA, naturally fit with the client-server model, the clients referencing (using) the services provided by the server. Service orientation, and therefore the ECOA, goes on a step extra, in that a component can be a client (service user) to one or more other components, whilst simultaneously being a *server* (service provider) to others.

This document presents the principal differences between the "*GimmeGimmeGimme*" and "*GimmeGimmeGimmePP*" client-server examples. It is assumed that the reader is conversant with the ECOA Architecture Specification (ref.[1]), the process of defining and declaring ECOA Assemblies, ASCs (components), Modules, and deployments in XML, using code generation to produce Module framework (stub) code units and ECOA Container and Platform code, and of course with the "*GimmeGimmeGimme*" example itself.

Aims

This ECOA "*GimmeGimmeGimmePP*" client-server example is intended to demonstrate, as C++, a minimum effort example of data distribution from a single data server to multiple data-accessing clients. Hence the example's name – "Gimme data (and lots of it) – and in C++".

ECOA Features Exhibited

- Composition of an ECOA Assembly of multiple ECOA ASCs (components).
- Contention-free resource sharing within an ECOA Assembly.
- Multiple cooperating ECOA Protection Domains.
- Use of the ECOA Logical Interface (ELI) between ECOA Protection Domains.
- Use of the ECOA runtime logging API.
- ASC (component) implementation in C++.

Design and Definition

The main point of this example is to repeat the "*GimmeGimmeGimme*" example but using C++ code. I won't therefore repeat all the design and capture-as-ECOA XML descriptions – it is exactly the same

as described in ref.[4]. With one exception. In the Server and Client Component Implementation XMLs (*Server.impl.xml* and *Client.impl.xml* respectively) instead of declaring the Module Implementation language as "C" it is "C++". That is:

```
<moduleImplementation name="modServer" moduleType="modServer_t" language="C++" />
```

and:

```
<moduleImplementation name="modClient" moduleType="modClient_t" language="C++" />
```

Implementation

The functional behaviour of the example is the same as for "*GimmeGimmeGimme*" so the implementation code is the same – except in C++ syntax. So as one example, the *Tick* ECOA Event Module Operation becomes, in C++:

```
void Module::Tick__received ()
{
    ECOA::int32 IAm;
    ECOA::return_status errc;
    sql::string requestStmt, responseData;
    //
    this->container->get_Id_value( /*&*/IAm );
    //
    requestStmt.current_size = sprintf( requestStmt.data,
                    "select * from iTable where %c = %d;",
                    (IAm==1?'A':'B'), this->user.Count++ );
    printf( "Request: %s\n", requestStmt.data );
    errc = this->container->Gimme__request_sync( requestStmt, responseData );
    if( errc != ECOA::return_status::OK ){
        printf( "Gimme request failed with errc = %ld\n", (long)errc );
    }else{
        // zero terminate responseData for printf...
        responseData.data[responseData.current_size] = '\000'
        printf( "Response: %s\n", responseData.data );
    }
}
```

Comparison with the C implementation of this Module Operation in ref.[4] illustrates, for instance, that:

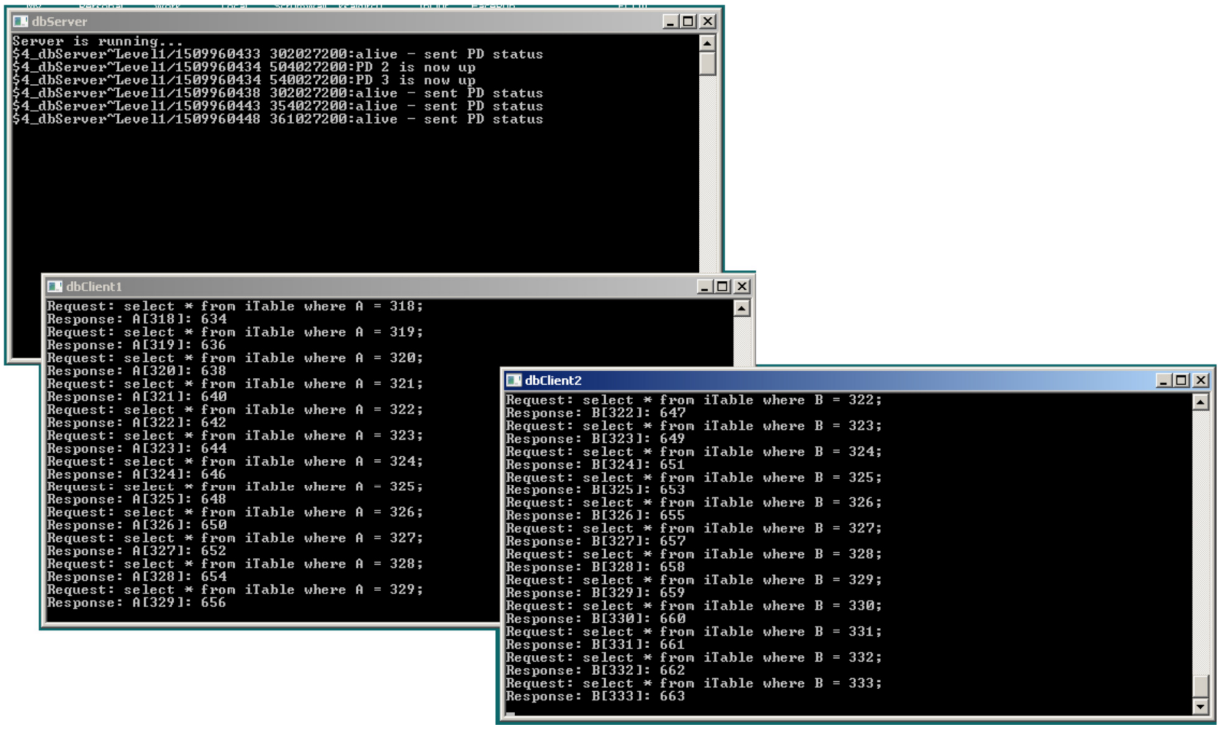
- There is no "*context*" parameter to the Operation. The "*context*" in C++ implementations is the object itself – referenced by "*this*". Hence the "User Context" variable "*Count*" is accessed using "*this->user.Count*" (rather than "*context->user.Count*");
- ECOA types are referred to using C++ syntax qualifiers: e.g. "*ECOA::int32*" rather than "*ECOA__int32*";
- Constants likewise are qualified: e.g. "*ECOA::return_status::OK*" rather than "*ECOA__return_status__OK*";
- The ECOA Container Interface is implemented through object methods, as in:

```
this->container->Gimme__request_sync(...);
```

Program Output

When the ECOA "*GimmeGimmeGimmePP*" Assembly is built and run (in a single Node deployment), an output similar to Figure 1 should be achieved, which (of course) should look just the same as with "*GimmeGimmeGimme*" from ref.[4]!

Figure 1 ECOA "*GimmeGimmeGimmePP*" in Execution



References

1	European Component Oriented Architecture (ECOA) Collaboration Programme: Architecture Specification (Parts 1 to 11) "ECOA" is a registered trade mark.
2	Client-server model *https://en.wikipedia.org/wiki/Client%E2%80%93server_model*
3	Gimme! Gimme! Gimme! (A Man After Midnight) ABBA, ©1979
4	ECOA Examples: *GimmeGimmeGimme* BAE Systems (Operations) Ltd., Electronic Systems.