***The ObjectWeb Consortium***

Tutorial

# Getting Started with OpenCCM
# -
# Building a CORBA Components Application

**AUTHORS:**

Areski Flissi (CNRS-LIFL)

**CONTRIBUTORS:**

Philippe Merle (INRIA)

Released: November 20, 2002

Status: Final Draft

Version: 1.0

# TABLE OF CONTENTS

**1 INTRODUCTION ..... 4**

1.1 GOALS ..... 4
1.2 TARGET AUDIENCE ..... 4
1.3 DOCUMENT CONVENTION ..... 4

**2 DESIGNING THE CORBA COMPONENTS APPLICATION ..... 5**

2.1 BUILDING THE APPLICATION BY ASSEMBLING CORBA COMPONENTS ..... 5
2.2 OMG IDL3 FILE STEPS TO DESIGN THE APPLICATION : ..... 6
2.3 THE SERVER COMPONENT ..... 7
2.4 THE CLIENT COMPONENT ..... 7
2.5 THE CONSUMER COMPONENT ..... 8

**3 COMPILATION AND GENERATION CHAIN FOR THE DEMO3 EXAMPLE... 9**

3.1 LOADING THE OPENCCM ENVIRONMENT ..... 9
3.2 START THE OPENCCM'S OMG IDL3 REPOSITORY (NAMED IR3) ..... 9
3.3 CHECKING THE DEMO3.IDL3 FILE ..... 9
3.4 FEEDING THE DEMO3.IDL3 FILE INTO THE OPENCCM'S IR3 ..... 10
3.5 GENERATING EQUIVALENT OMG IDL 2.4 MAPPING FOR DEMO3 DEMO3.IDL ..... 10
3.6 OMG IDL MAPPING RULES ..... 10
*3.6.1 Client-side OMG IDL Mapping rules ..... 10*
*3.6.2 Server-side OMG IDL Mapping Rules ..... 14*
3.7 GENERATING THE JAVA OPENCCM SKELETONS FOR DEMO3 ..... 19
3.8 GENERATING JAVA CORBA 2 STUBS ..... 20
3.9 IMPLEMENTING THE CLIENT / SERVER – PRODUCER / CONSUMER EXAMPLE ..... 20
3.10 COMPILING GENERATED JAVA CORBA 2 STUBS, GENERATED JAVA OPENCCM SKELETONS, ALL JAVA IMPLEMENTATION SOURCES AND BUILDING ARCHIVE DEMO3.JAR ..... 26

**4 EXECUTION CHAIN FOR THE DEMO3 EXAMPLE ..... 27**

4.1 INSTALLING THE OPENCCM'S CONFIGURATION REPOSITORY ..... 27
4.2 STARTING THE NAME SERVICE USED BY THE OPENCCM'S EXECUTION CHAIN ..... 27
4.3 START JAVA COMPONENT SERVERS FOR DEMO3 NAMED : COMPONENTSERVER1 AND COMPONENTSERVER2 ..... 27
4.4 STARTING A JAVA VIRTUAL MACHINE TO START DEMO3 ..... 28
4.5 STOP THE DEMO3 ..... 29

**5 XML DESCRIPTORS FOR PACKAGING, ASSEMBLING AND DEPLOYING THE APPLICATION ..... 30**

5.1 SOFTWARE PACKAGE DESCRIPTOR ..... 30
5.2 PROPERTY FILE DESCRIPTOR ..... 31
5.3 COMPONENT ASSEMBLY DESCRIPTOR ..... 32

# TABLE OF FIGURES

FIGURE 1 – CORBA COMPONENTS OF THE APPLICATION ..... 5
FIGURE 2 – SERVER COMPONENT AND ITS HOMES ..... 7
FIGURE 3 – CLIENT COMPONENT AND ITS HOMES ..... 8
FIGURE 4 – CONSUMER COMPONENT AND ITS HOMES ..... 8
FIGURE 5 – CORBA COMPONENTS COMPILATION CHAIN FROM CLIENT VIEW POINT ..... 11
FIGURE 6 – CLIENT-SIDE OMG IDL MAPPING RULES ..... 14
FIGURE 7 – CORBA COMPONENTS COMPILATION CHAIN FROM SERVER VIEW POINT ..... 14
FIGURE 8 – SERVER-SIDE OMG IDL MAPPING RULES ..... 18
FIGURE 9 – INTERCONNECTIONS BETWEEN CLIENT, SERVER/PRODUCER AND CONSUMER COMPONENTS ..... 21
FIGURE 10 – THE APPLICATION AT WORK ..... 29
FIGURE 11 – USING XML DESCRIPTORS FOR PACKAGING, ASSEMBLING AND DEPLOYING THE APPLICATION ..... 30

# 1 INTRODUCTION

This document shows how to build and run a simple CORBA Components application with the OpenCCM platform. Moreover, this tutorial covers all the CORBA Components Specification features currently implemented in the OpenCCM platform.

The application shown in this tutorial is composed of a set of interconnected CORBA components deployed in several distributed application servers. The application illustrates both client-server and producer-consumer relationships between CORBA components.

The full source code of the presented application is available in the demo/demo3/ directory of the OpenCCM distribution.

## 1.1 Goals

This tutorial illustrates how:

- to define OMG IDL component-oriented interfaces,
- to compile OMG IDL with the OpenCCM compilation chain,
- to implement CORBA components,
- to compile all Java source files,
- to use the OpenCCM execution chain,
- to run the application, and
- to write CORBA Components packaging and assembling XML descriptors.

## 1.2 Target audience

The target audience for this tutorial includes all OpenCCM users and developers.

## 1.3 Document Convention

Description: Times New Roman:12

Example or source code: `Courier New:10`

# 2 DESIGNING THE CORBA COMPONENTS APPLICATION

The target application is composed of a set of client components connected to a server component, itself connected to a set of consumer components. The client components could synchronously call a service provided by the server component. Each time the service is called, the server component notifies/publishes an asynchronous event to all the connected consumer components. Components are created by simple component homes or managers with primary keys.

## 2.1 Building the application by assembling CORBA Components

With the CORBA Components Specification and the OpenCCM platform, an application is generally built as an assembly of interconnected CORBA components deployed on several distributed application servers.

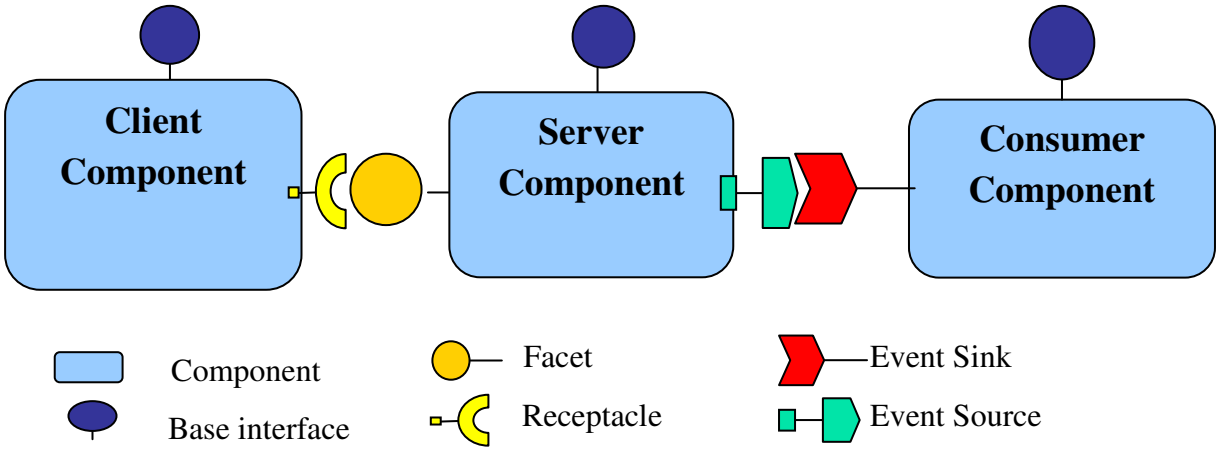



**Figure 1 – CORBA Components of the Application**

A CORBA component is a unit encapsulating business code implementation and exposing public component features through clearly well identified interfaces. By default, each component has a base interface exposing business specific configurable properties (like a colour and a title for a window component) and providing a set of generic control operations defined in the inherited Components::CCMObject interface.

Depending of its business, a CORBA component could provide several facet interfaces to other CORBA components. Each facet interface captures a distinct functional point of view on the component features. A facet interface could contain a set of standard OMD IDL operations and attributes. For instance in Figure 1, the Server component provides a facet interface used by Client components. Complementary, a CORBA component could clearly identify what interfaces he requires to run correctly. These required interfaces are named receptacles and represent connection points of a component. For instance in Figure 1, the Client component has a receptacle which must be connected to the Service interface.

On the other side, a CORBA component could provide event-oriented interfaces (named event sinks) to consume events. In Figure 1, the Consumer component has an event sink interface. Complementary, a CORBA component could clearly identify which events he produces

through event sources. In Figure 1, the Server component has an event source which must be connected to some event sinks.

Then a CORBA architect could build or assemble its applications by simply connecting component facets to component receptacles and component event sinks to component event sources.

## 2.2 OMG IDL3 file steps to design the application :

- To have access to OMG IDL definitions contained into the CCM's Components module

```
import Components;
```

- Prefixing generated Java mapping classes

```
typeprefix demo3 "ccm.objectweb.org";
```

- The Service interface provided by the Server component and used by its Client components

```
interface Service
{
    void display(in string text);
};
```

- The Event valuetype published by the Server component and consumed by its Consumer components

```
eventtype TextEvent
{
    /** Just contains a string. */
    public string text;
};
```

- A base type for named components

```
component NamedComponent
{
    /** The identifier name property. */
    attribute string name;
};
```

- The primary key to identify components

```
valuetype NamePrimaryKey : ::Components::PrimaryKeyBase
```

```
{
    /** Just a string name. */
    public /*string*/ long name;
};
```

## 2.3 The Server component

```
component Server : NamedComponent
{
    /** Provides a Service to its
     * Client components. */
    provides Service the_service;

    /** Publishes Events to its
     * Consumer components. */
    publishes TextEvent to_consumers;
};
```

- The simple home for instantiating Server components

```
home ServerHome manages Server
{
};
home ServerManager manages Server
        primarykey NamePrimaryKey
{
    /** To create a new Server
     * identified by the name. */
    factory create_server(in string
                          name);
    /** To find a Server identified
     * by the name. */
    finder find_server(in string name);
};
```

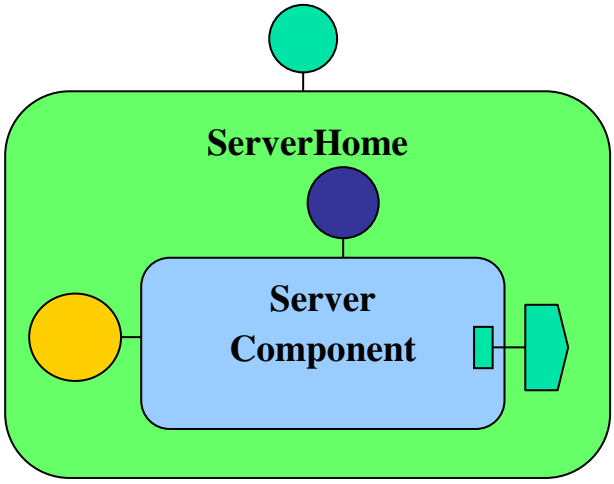



**Figure 2 – Server Component and its Homes**

## 2.4 The Client Component

- The Client component type

```
component Client : NamedComponent
{
    // Uses the service provided by the
    // Server component.
    uses Service the_service;
};
```

- The home for managing Client components

```
home ClientHome manages Client
{
};
home ClientManager manages Client
          primarykey NamePrimaryKey
{
    /** To create a new Client identified
     * by the name. */
    factory create_client(in string
                               name);

    /** To find a Client identified by
     * the name. */
    finder find_client(in string name);
};
```

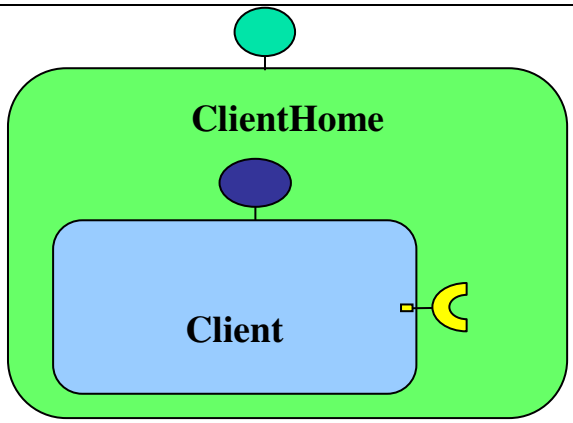



**Figure 3 – Client Component and its Homes**

## 2.5 The Consumer Component

- The Consumer component type

```
component Consumer : NamedComponent
{
    // Consumes Events published
    // by Server components
    consumes TextEvent from_servers;
};
```

- The home for Client components

```
home ConsumerManager manages Consumer
                primarykey NamePrimaryKey
{
    /** To create a new Consumer
     *  identified by the name. */
    factory create_consumer(in string
                                name);

    /** To find a Consumer identified
     *  by the name. */
    finder find_consumer(in string name);
};
```

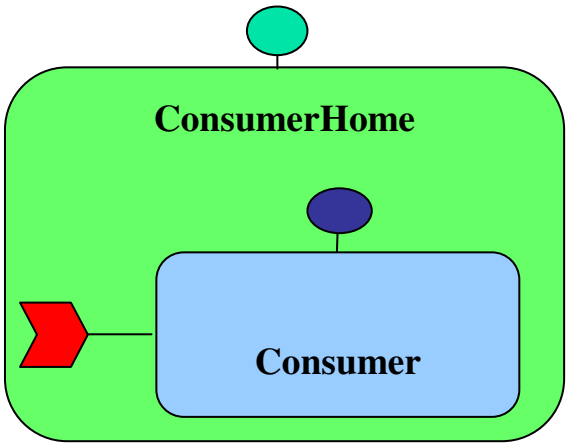



**Figure 4 – Consumer Component and its Homes**

# 3 COMPILATION AND GENERATION CHAIN FOR THE DEMO3 EXAMPLE

Assuming OpenCCM is compiled and installed in C:\OpenCCM with ORBacus-4.1 under Windows NT:

## 3.1 Loading the OpenCCM environment

```
C:\OpenCCM>call ORBacus-4.1\bin\envi_OpenCCM.bat
```

## 3.2 Start the OpenCCM's OMG IDL3 Repository (named IR3)

```
C:\OpenCCM\demo\demo3>ir3_start
The OpenCCM's OMG IDL3 Repository will be started.
Creating the C:\OpenCCM\ORBacus-4.1\OpenCCM_CONFIG_DIR directory.
Launching the OpenCCM's IR3.
Feeding the OpenCCM's IR3 with the IFR_3_0.idl file.
OpenCCM's OMG IDL 3.0 Compiler 0.5.0: Reading from file C:\OpenCCM\ORBacus-
4.1\idl\IFR_3_0.idl...
OpenCCM's OMG IDL 3.0 Compiler 0.5.0: Preprocessing file
C:\OpenCCM\ORBacus-4.1\idl\IFR_3_0.idl...
OpenCCM's OMG IDL 3.0 Compiler 0.5.0: File C:\OpenCCM\ORBacus-
4.1\idl\IFR_3_0.idl preprocessed
OpenCCM's OMG IDL 3.0 Compiler 0.5.0: Feeding the Interface Repository ...
OpenCCM's OMG IDL 3.0 Compiler 0.5.0: Compilation completed: 0 warnings.
Feeding the OpenCCM's IR3 with the Components.idl file.
OpenCCM's OMG IDL 3.0 Compiler 0.5.0: Reading from file C:\OpenCCM\ORBacus-
4.1\idl\Components.idl...
OpenCCM's OMG IDL 3.0 Compiler 0.5.0: Preprocessing file
C:\OpenCCM\ORBacus-4.1\idl\Components.idl...
OpenCCM's OMG IDL 3.0 Compiler 0.5.0: File C:\OpenCCM\ORBacus-
4.1\idl\Components.idl preprocessed
OpenCCM's OMG IDL 3.0 Compiler 0.5.0: Feeding the Interface Repository ...
OpenCCM's OMG IDL 3.0 Compiler 0.5.0: Compilation completed: 0 warnings.
The OpenCCM's OMG IDL3 Repository is started.
C:\OpenCCM\demo\demo3>
```

## 3.3 Checking the demo3.idl3 file

```
C:\OpenCCM\demo\demo3>idl3_check demo3.idl3
OpenCCM's OMG IDL 3.0 Compiler 0.5.0: Reading from file demo3.idl3...
OpenCCM's OMG IDL 3.0 Compiler 0.5.0: Preprocessing file demo3.idl3...
OpenCCM's OMG IDL 3.0 Compiler 0.5.0: File demo3.idl3 preprocessed
OpenCCM's OMG IDL 3.0 Compiler 0.5.0: Feeding the Interface Repository ...
OpenCCM's OMG IDL 3.0 Compiler 0.5.0: Compilation completed: 0 warnings.
```

## 3.4 Feeding the demo3.idl3 file into the OpenCCM's IR3

```
C:\OpenCCM\demo\demo3>ir3_feed demo3.idl3
OpenCCM's OMG IDL 3.0 Compiler 0.5.0: Reading from file demo3.idl3...
OpenCCM's OMG IDL 3.0 Compiler 0.5.0: Preprocessing file demo3.idl3...
OpenCCM's OMG IDL 3.0 Compiler 0.5.0: File demo3.idl3 preprocessed
OpenCCM's OMG IDL 3.0 Compiler 0.5.0: Feeding the Interface Repository ...
OpenCCM's OMG IDL 3.0 Compiler 0.5.0: Compilation completed: 0 warnings.

C:\OpenCCM\demo\demo3>
```

## 3.5 Generating equivalent OMG IDL 2.4 mapping for demo3 demo3.idl

The ir3_idl2 script allows one to generate the OMG IDL 2.4 CCM's mapping associated to an OpenCCM's IR3 object (demo3.idl) :

```
C:\OpenCCM\demo\demo3>ir3_idl2 demo3
```

Add –o filename option to produce an output file and –i option to add the #include statement, ie :

```
C:\OpenCCM\demo\demo3>ir3_idl2 –i Components.idl –o demo3.idl demo3
```

In our case, *"-i Components.idl"* produces the *"#include Components.idl"* statement in the demo3.idl file

## 3.6 OMG IDL Mapping Rules

### 3.6.1 Client-side OMG IDL Mapping rules

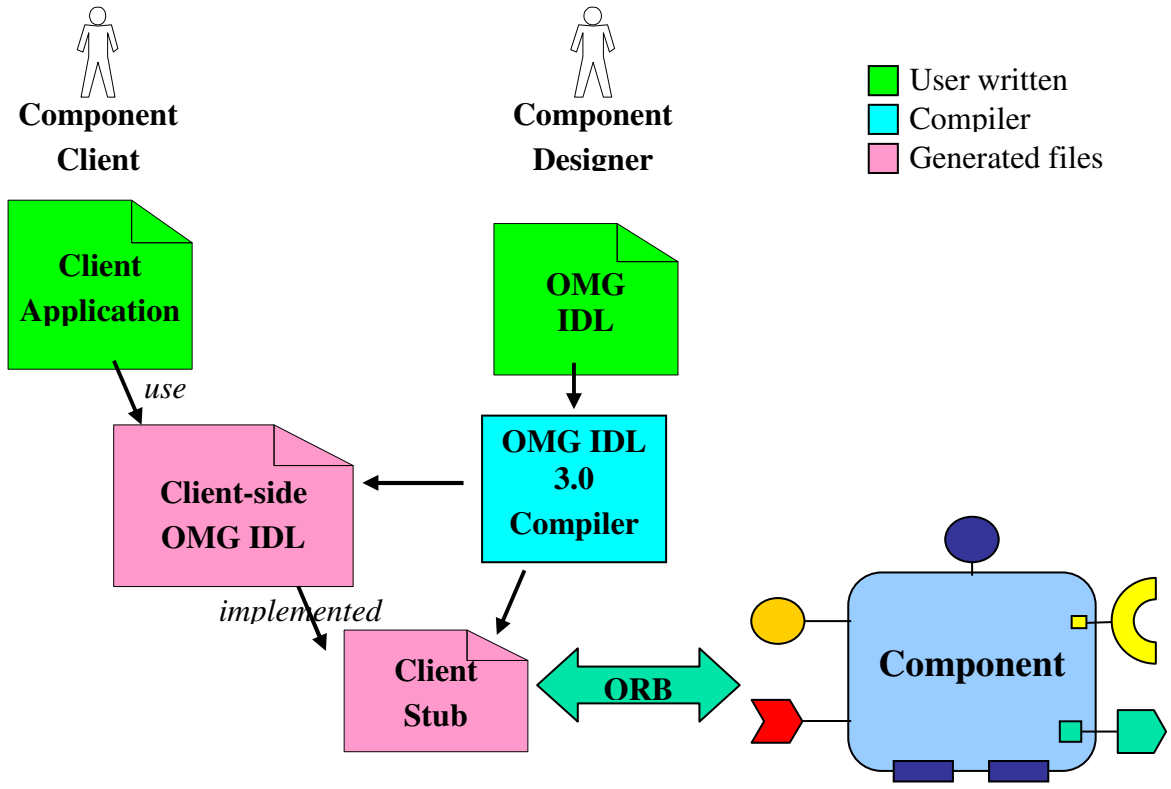



**Figure 5 – CORBA Components Compilation Chain from Client View Point**

General rules for the Client-side OMG IDL mapping rules :

- A component type is mapped to an interface inheriting from **Components::CCMObject**
- Facets and event sinks are mapped to an operation for obtaining the associated reference
- Receptacles are mapped to operations for connecting, disconnecting, and getting the associated reference(s)
- Event sources are mapped to operations for subscribing and unsubscribing to produced events
- An event type is mapped to
  - + A value type inheriting from **Components::EventBase**
  - + A consumer interface inheriting from **Components::EventConsumerBase**
- A home type is mapped to three interfaces
  - + One for explicit operations user-defined inheriting from **Components::CCMHome**
  - + One for implicit operations generated
  - + One inheriting from both previous interfaces

For our Client/Server-Producer/Consumer application, the main client-side OMG IDL mapping rules are :

<table>
<tr><th>OMG IDL3</th><th>OMG IDL 2.4 mapping</th></tr>
<tr><td>

```
eventtype TextEvent
{
    /** Just contains a string. */
    public string text;
};
```

</td><td>

```
valuetype TextEvent : ::Components::EventBase
{
   public string text;
};
interface TextEventConsumer :
            ::Components::EventConsumerBase
{
   void push_TextEvent(in ::demo3::TextEvent
                              the_textevent);
};
```

</td></tr>
<tr><td>

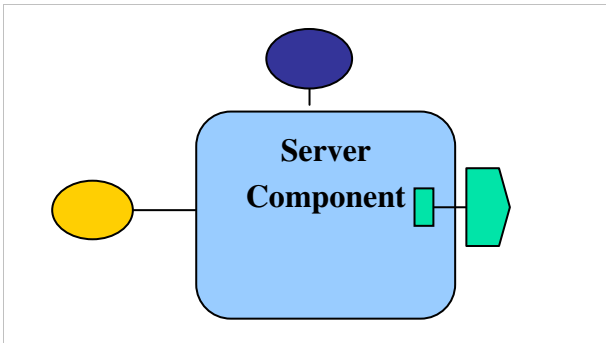
```
interface Service
{
   void display(in string text);
};
```

</td><td>

```
interface Service
{
   void display(in string text);
};
```

</td></tr>
<tr><td>

```
component Server : NamedComponent
{
   provides Service the_service;
   publishes TextEvent to_consumers;
};
```

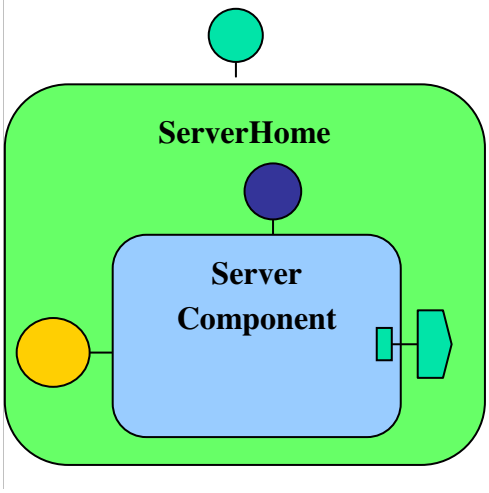



</td><td>

```
interface Server : ::demo3::NamedComponent
{
   ::demo3::Service provide_the_service();

   ::Components::Cookie
    subscribe_to_consumers(in
            ::demo3::TextEventConsumer consumer);
    ::demo3::TextEventConsumer
    unsubscribe_to_consumers(in
                              ::Components::Cookie ck);
};
```

</td></tr>
</table>

```
home ServerHome manages Server {};
```

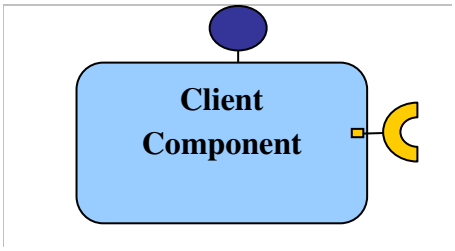



```
interface ServerHomeExplicit :
            ::Components::CCMHome
{
};
interface ServerHomeImplicit :
            ::Components::KeylessCCMHome
{
   ::demo3::Server create();
};
interface ServerHome :
            ::demo3::ServerHomeExplicit,
            ::demo3::ServerHomeImplicit
{
};
```

```
home ServerManager manages Server
            primarykey NamePrimaryKey
{
   factory create_server(in string name);
   finder find_server(in string name);
};
```

```
interface ServerManagerExplicit :
            ::Components::CCMHome
{
   ::demo3::Server create_server(in string name)
   ::demo3::Server find_server(in string name)
};
interface ServerManagerImplicit
{
   ::demo3::Server create(in
                        ::demo3::NamePrimaryKey key);
    ::demo3::Server find_by_primary_key(in
                        ::demo3::NamePrimaryKey key);
     void remove(in ::demo3::NamePrimaryKey key);
    ::demo3::NamePrimaryKey get_primary_key(in
                                ::demo3::Server comp);
};
interface ServerManager :
            ::demo3::ServerManagerExplicit,
           ::demo3::ServerManagerImplicit  { };
```

```
component Client : NamedComponent
{
   uses Service the_service;
};
```




```
interface Client : ::demo3::NamedComponent
{
  void connect_the_service (in
                        ::demo3::Service connexion);
  ::demo3::Service disconnect_the_service();
  ::demo3::Service get_connection_the_service();
};
```

**Figure 6 – Client-side OMG IDL mapping rules**

### 3.6.2 Server-side OMG IDL Mapping Rules

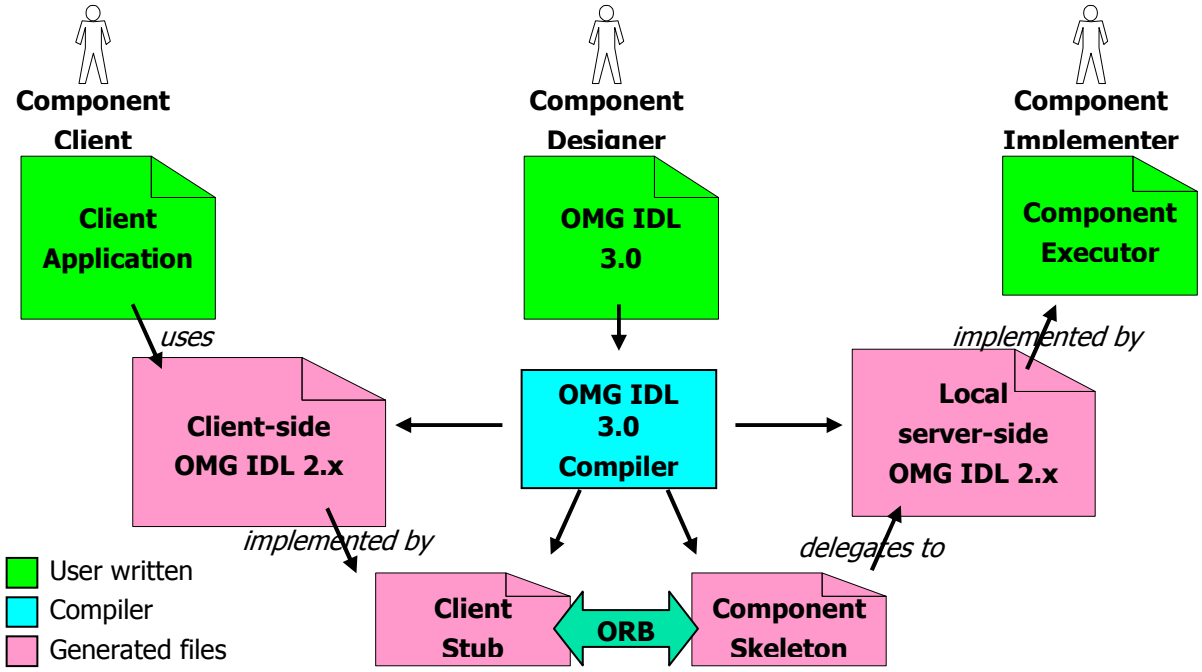


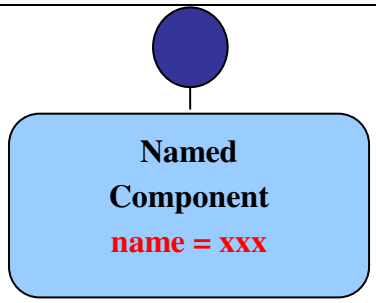

**Figure 7 – CORBA Components Compilation Chain from Server View Point**

General rules for the Server-side OMG IDL mapping rules :

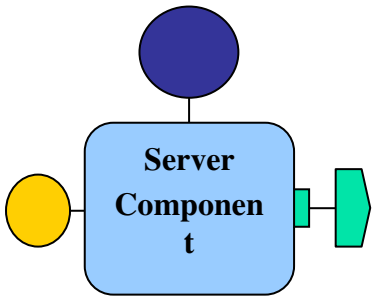
- A component type is mapped to three local interfaces
  - + The main component executor interface
    Inheriting from Components::EnterpriseComponent
  - + The monolithic component executor interface
    Operations to obtain facet executors and receive events
  - + The component specific context interface
    Operations to access component receptacles and event sources

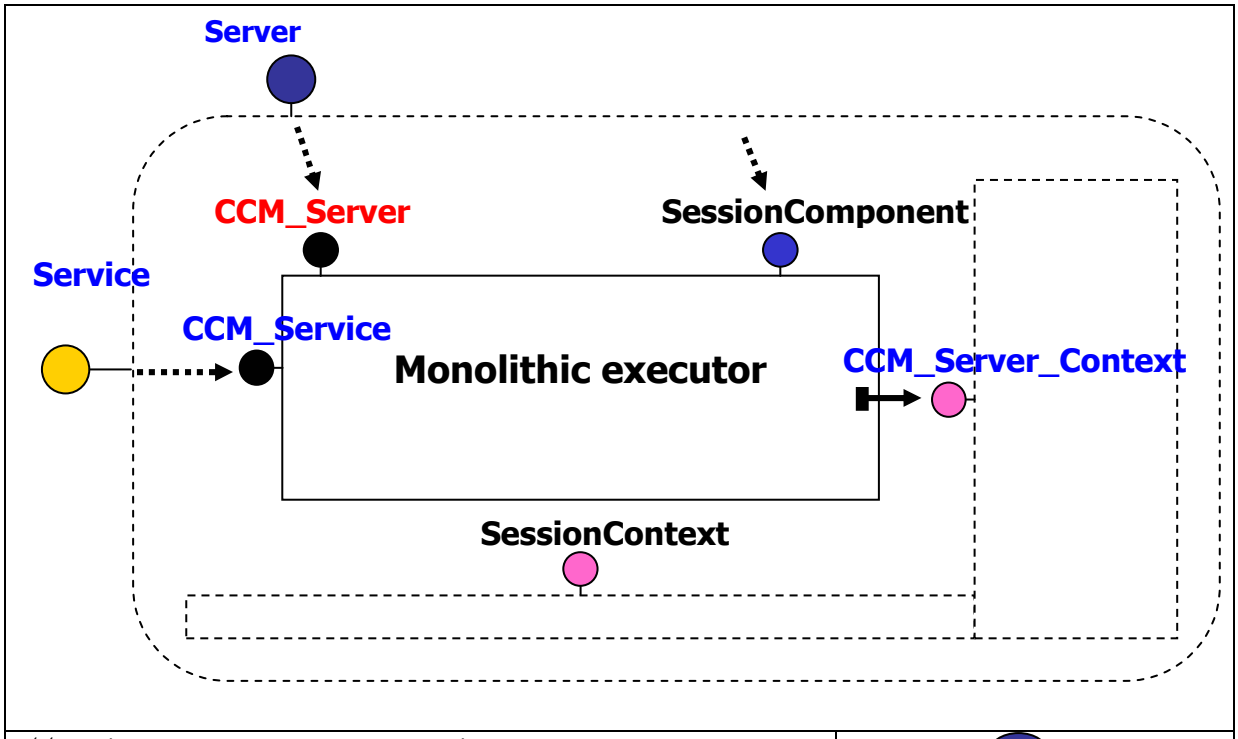
- A home type is mapped to three local interfaces
  - + One for explicit operations user-defined
    Inheriting from Components::HomeExecutorBase
  - + One for implicit operations generated
  - + One inheriting from both previous interfaces

```
// Main component executor interface
local interface CCM_Server_Executor :
               ::demo3::CCM_NamedComponent_Executor
{
};

// Monolithic component executor interface
local interface CCM_Server :
                ::demo3::CCM_Server_Executor
{
    ::demo3::CCM_Service get_the_service();
};

// Component-specific context interface.
local interface CCM_Server_Context :
               ::demo3::CCM_NamedComponent_Context
{
    void push_to_consumers(in ::demo3::TextEvent
                                event);
};
```

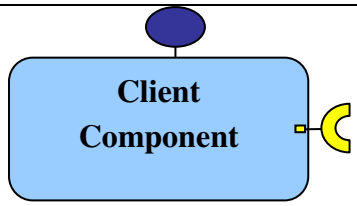


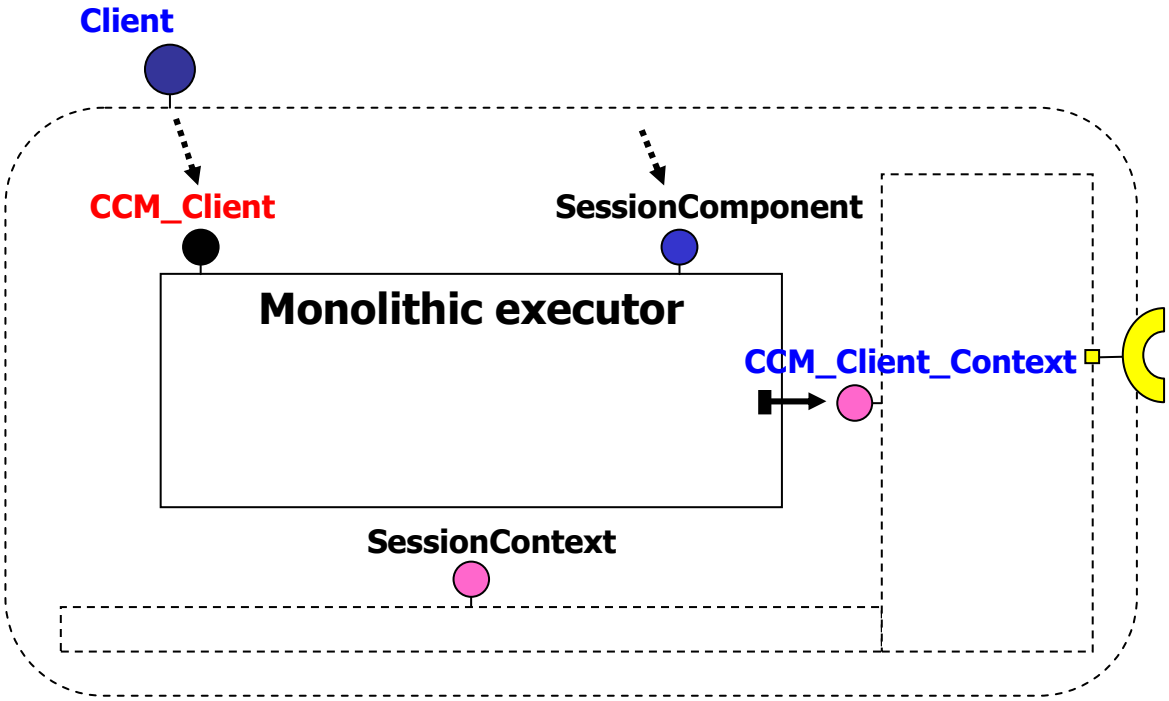

```
// Main component executor interface
local interface CCM_Server_Executor :
               ::demo3::CCM_NamedComponent_Executor
{
};

// Monolithic component executor interface
local interface CCM_Server :
                ::demo3::CCM_Server_Executor
{
    ::demo3::CCM_Service get_the_service();
};

// Component-specific context interface.
local interface CCM_Server_Context :
               ::demo3::CCM_NamedComponent_Context
{
    void push_to_consumers(in ::demo3::TextEvent
                              event);
};
```



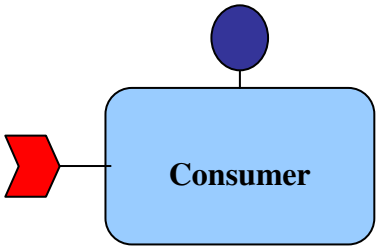



```
// Main component executor interface
local interface CCM_Client_Executor :
       ::demo3::CCM_NamedComponent_Executor
{
};

// Monolithic component executor interface
local interface CCM_Client :
                ::demo3::CCM_Client_Executor
{
};

// Component-specific context interface.
local interface CCM_Client_Context :
               ::demo3::CCM_NamedComponent_Context
{
    ::demo3::Service get_connection_the_service();
};
```






```
local interface CCM_TextEventConsumer
{
    void push(in ::demo3::TextEvent event);
};

// Main component executor interface
local interface CCM_Consumer_Executor :
               ::demo3::CCM_NamedComponent_Executor
{
};

// Monolithic component executor interface
local interface CCM_Consumer :
                ::demo3::CCM_Consumer_Executor
{
    void push_from_servers(in ::demo3::TextEvent
                           event);
};

// Component-specific context interface.
local interface CCM_Consumer_Context :
               ::demo3::CCM_NamedComponent_Context
{
};
```



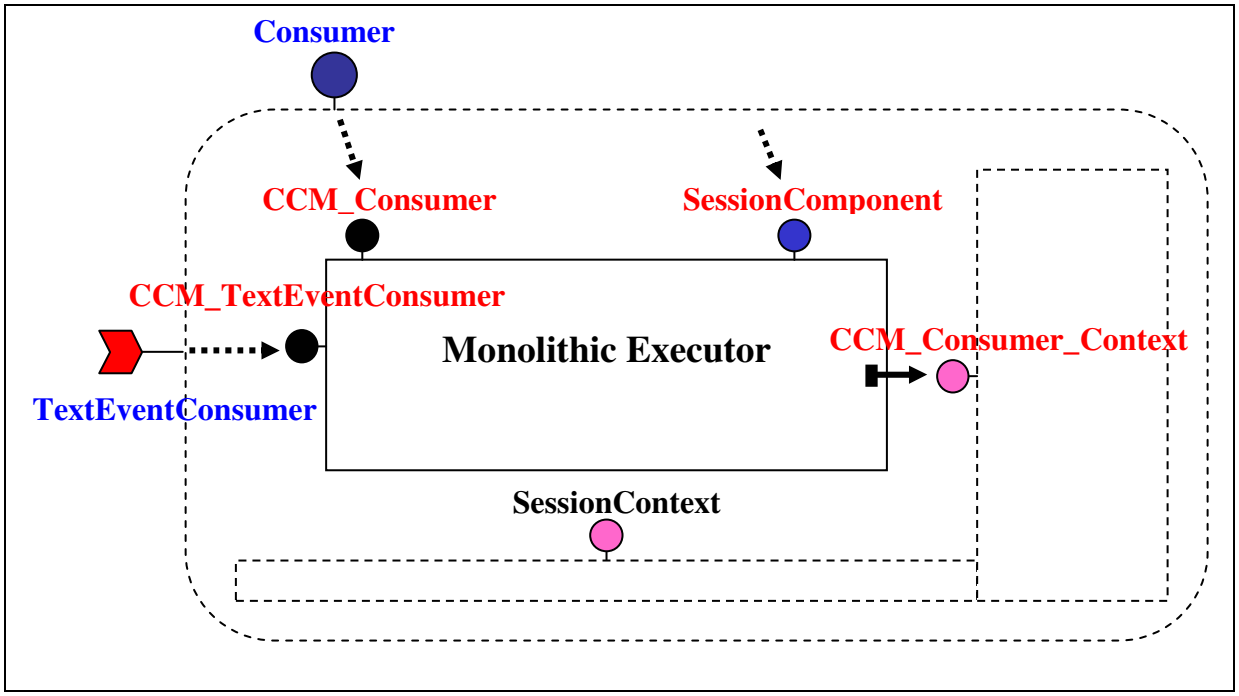



**Figure 8 – Server-side OMG IDL mapping rules**

## 3.7 Generating the Java OpenCCM skeletons for demo3

```
C:\OpenCCM\demo\demo3>ir3_java ::demo3
```

Files generated are:

```
ClientCCM.java
ClientHomeCCM.java
ClientHomeSkeletonInterceptor.java
ClientHomeStubInterceptor.java
ClientManagerCCM.java
ClientManagerSkeletonInterceptor.java
ClientManagerStubInterceptor.java
ClientMonolithicWrapper.java
ClientSkeletonInterceptor.java
ConsumerCCM.java
ConsumerHomeCCM.java
ConsumerHomeSkeletonInterceptor.java
ConsumerHomeStubInterceptor.java
ConsumerManagerCCM.java
ConsumerManagerSkeletonInterceptor.java
ConsumerManagerStubInterceptor.java
ConsumerMonolithicWrapper.java
ConsumerSkeletonInterceptor.java
NamedComponentCCM.java
NamedComponentMonolithicWrapper.java
NamedComponentSkeletonInterceptor.java
NamePrimaryKeyFactoryHelper.java
ServerCCM.java
ServerHomeCCM.java
ServerHomeSkeletonInterceptor.java
ServerHomeStubInterceptor.java
ServerManagerCCM.java
ServerManagerSkeletonInterceptor.java
ServerManagerStubInterceptor.java
ServerMonolithicWrapper.java
ServerSkeletonInterceptor.java
ServiceSkeletonInterceptor.java
ServiceStubInterceptor.java
TextEventConsumerSkeletonInterceptor.java
TextEventConsumerStubInterceptor.java
TextEventConsumerWrapper.java
TextEventFactoryHelper.java
```

Skeleton files

- For each component definition, a Java skeleton file is generated (ClientCCM.java, ConsumerCCM.java, ServerCCM.java)
- For each home definition, a Java skeleton file associated is generated (ServerHomeCCM.java...)

Helper files

- Event type definition : TextEventFactoryHelper.java is generated. This file provides facilities to register a factory for the equivalent OMG IDL 2.4 value type, to narrowcast a base event type into this type and to create an any from an event of this type.
- Valuetype NamedPrimaryKey : NamePrimaryKeyFactoryHelper.java is generated

Internally used files

- For Client, Server and Consumer components, ClientSkeletonInterceptor.java, ServerSkeletonInterceptor.java, ConsumerSkeletonInterceptor.java and *MonotlithicWrapper.java are generated
- For each home definition, two interceptor files are generated : ClientHomeSkeletonInterceptor.java, ClientHomeStubInterceptor.java in the case of Client home definition

## 3.8 Generating Java CORBA 2 stubs

```
C:\OpenCCM\demo\demo3>jidl --auto-package --tie -I. -I../../ORBacus-4.1/idl
--output-dir generated/stubs demo3.idl
```

Using jidl compiler for ORBacus-4.1, --tie option allows to generate tie classes, -I option to include idl files

This generate stubs for demo3 in demo3\generated\stubs directory.

## 3.9 Implementing the Client / Server – Producer / Consumer example

Now we have to implement this example by writing Java implementation files

Note that we implement now functionnal parts of the application (non-functionnal parts are implemented by the skeletons). Let's have a look at the implementation files we must write :

```
ClientHomeImpl.java
ClientImpl.java
ConsumerHomeImpl.java
ConsumerImpl.java
Demo3.java
ServerHomeImpl.java
ServerImpl.java
TextEventDefaultFactory.java
TextEventImpl.java
```

**Demo3.java** is the bootstrap of the application, here we have to

- Initialize the ORB
- Obtain the Name Service by using *orb.resolve_initial_references("NameService")* method
- Obtain component servers ComponentServer1 and ComponentServer2
- Obtain the container homes
- Instanciate a container on server1 and server2
- Install homes for Client, Server and Consumer
- Create components with *create()* method of homes : here we have three clients, three consumers and one server - producer
- Configure components using the *name("...")* method, for each client, consumer and server (see component implementation for more details...)
- Connect each client and consumer to server by using *connect_the_service(the_service)* for each client and *subscribe_to_consumers(...)* for consumers
- Call the *configuration_complete()* method

The next figure shows the interconnections between the instanciated components:

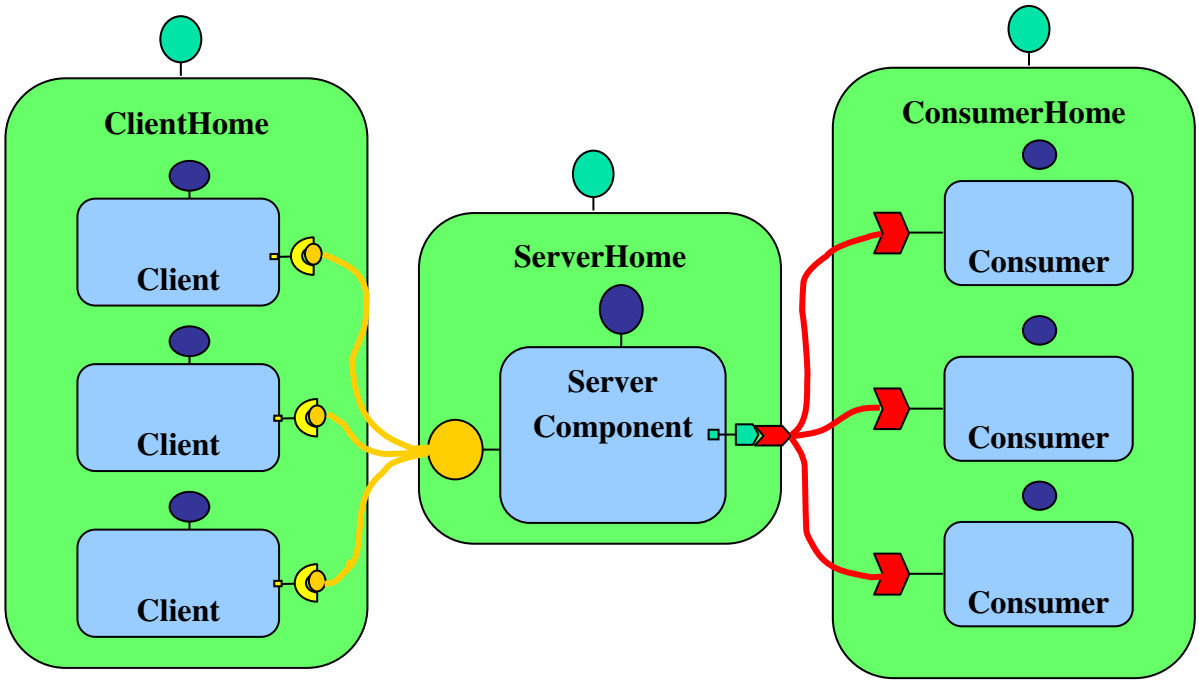



**Figure 9 – Interconnections between client, server/producer and consumer components**

The implementation of connections of client and consumer to server components looks like:

....

```
// component Server provide the service facet
Service the_service = s.provide_the_service();
// components Client connect to the Service facet
c1.connect_the_service(the_service);
```

```
c2.connect_the_service(the_service);
c3.connect_the_service(the_service);
// Consumers consume the events published by server
s.subscribe_to_consumers(cs1.get_consumer_from_servers());
s.subscribe_to_consumers(cs2.get_consumer_from_servers());
s.subscribe_to_consumers(cs3.get_consumer_from_servers());
...
```

**ClientImpl.java :**

- Instanciates, constructs and shows the GUI
- Perform the action when the button is clicked by calling the display service *service.display(...)* (Service interface operation)

```
...
    /**
     * When the button is selectionned.
     *
     * @param e The associate ActionEvent.
     */
    public void
    actionPerformed(java.awt.event.ActionEvent e)
    {
        // Obtain the object reference associated to the
        // 'the_service' receptacle.
        Service service = the_context_.get_connection_the_service();

        // Check if the connection is available.
        if(service == null)
        {
            System.err.println("The demo3::Client::the_service receptacle
is not set!");
         return;
        }

        // Calls the display service.
        service.display(name_ + ":" + text_.getText());
    }
...
```

**ServerImpl.java**

- Instanciates, constructs and shows the GUI
- Implements mutator and accessor methods for attribute name
- Implements the display method for the Service interface by displaying string text
- Push events to consumers *push_to_consumers(... )*

```
...
    /**
```

```
     * The display operation of the demo3::Service interface.
     *
     * @param text The text to display.
     */
    public void
    display(String text)
    {
        // Puts the text into the text area.
        textArea_.append(text + "\n");

        // Pushes an event to all connected consumers.
        the_context_.push_to_consumers( new TextEventImpl(text) );
    }
...
```

**ConsumerImpl.java :**

- Instanciates, constructs and shows the GUI
- Implements reception of events published by server components *push_from_server(...)*

```
...
    /**
     * To receive demo3::Event events published by demo3::Server components.
     *
     * @param event The received event.
     */
    public void
    push_from_servers(TextEvent event)
    {
        push(event);
    }

     /**
     * To receive demo3::Event events published by demo3::Server components.
     *
     * @param event The received event.
     */
    public void
    push(TextEvent event)
    {
        // Put the text into the text area.
        textArea_.append(event.text + "\n");
    }
...
```

The *push(...)* method simply display text on consumer's GUI.

**ClientHomeImpl.java**

This class inherits from the local CCM_ClientHome interface generated by the OpenCCM's IDL3 to IDL2 mapping generator. It implements the *create_home()* method called by the OpenCCM Component Server to create a home instance

```
...
    public static org.omg.Components.HomeExecutorBase
    create_home()
    {
        return new ClientHomeImpl();
    }
...
```

**ConsumerHomeImpl.java**

- Implements the *create_home()* method to create a home instance
- Register the TextEvent valuetype factory to the ORB

```
...
    public static org.omg.Components.HomeExecutorBase
    create_home()
    {
        return new ConsumerHomeImpl();
    }

    // Executed once at the loading of this class
    // by the OpenCCM Component Server.
    static
    {
        // Required to register the TextEvent valuetype factory to the ORB.
        TextEventDefaultFactory.register();
    }
...
```

**ServerHomeImpl.java**

- Implements the *create_home()* method to create a home instance
- Register the TextEvent valuetype factory to the ORB

```
    public static org.omg.Components.HomeExecutorBase
    create_home()
    {
        return new ServerHomeImpl();
    }

    // Executed once at the loading of this class
    // by the OpenCCM Component Server.
    static
    {
        // Required to register the Event valuetype factory to the ORB.
        TextEventDefaultFactory.register();
    }
```

### Implementation of the Event valuetype factory :

**TextEventDefaultFactory.java**

```
...
    /**
     * Register the valuetype factory to the ORB.
     */
    public static void
    register()
    {
        TextEventFactoryHelper.register(new TextEventDefaultFactory());
    }
...
```

### Implementation of the Event valuetype : TextEventImpl.java

```
public class TextEventImpl
     extends TextEvent
{

    /** The default constructor. */
    public
    TextEventImpl()
    {
        text = "";
    }

    /**
     * The constructor.
     *
     * @param t A text.
     */
    public
    TextEventImpl(String t)
    {
        text = t;
    }
}
```

## 3.10 Compiling generated Java CORBA 2 stubs, generated Java OpenCCM skeletons, all Java implementation sources and building archive demo3.jar

```
idl2java:
     [echo]   Compiling the OMG IDL file demo/demo3/demo3.idl

compile_stubs:
     [echo] Compiling generated Java CORBA 2 stubs.
    [javac] Compiling 302 source files to
C:\OpenCCM\demo\demo3\generated\classes
    [javac] Note: Some input files use or override a deprecated API.
    [javac] Note: Recompile with -deprecation for details.

compile_skeletons:
     [echo] Compiling generated Java OpenCCM skeletons.
    [javac] Compiling 37 source files to
C:\OpenCCM\demo\demo3\generated\classes
    [javac] Note: Some input files use or override a deprecated API.
    [javac] Note: Recompile with -deprecation for details.

compile_implementations:
     [echo] Compiling all Java implementation sources.
    [javac] Compiling 9 source files to
C:\OpenCCM\demo\demo3\generated\classes
    [javac] Note: Some input files use or override a deprecated API.
    [javac] Note: Recompile with -deprecation for details.

build_archive:
     [echo] Building demo/demo3/archives/demo3.jar
      [jar] Building jar: C:\OpenCCM\demo\demo3\archives\demo3.jar
```

# 4 EXECUTION CHAIN FOR THE DEMO3 EXAMPLE

## 4.1 Installing the OpenCCM's Configuration Repository

```
C:\OpenCCM\demo\demo3>ccm_install
The OpenCCM Platform will be installed.
Creating the C:\OpenCCM\ORBacus-4.1\OpenCCM_CONFIG_DIR directory.
Creating the C:\OpenCCM\ORBacus-4.1\OpenCCM_CONFIG_DIR\ComponentServers
directory.
The OpenCCM Platform is installed.
```

## 4.2 Starting the Name Service used by the OpenCCM's Execution Chain

```
C:\OpenCCM\demo\demo3>ns_start
The Name Service will be started.
Launching the ORBacus Name Service.
The Name Service is started.
```

## 4.3 Start Java Component Servers for demo3 named : ComponentServer1 and ComponentServer2

```
C:\OpenCCM\demo\demo3>jcs_start ComponentServer1
The OpenCCM's Java Component Server ComponentServer1 will be started.
Creating the C:\OpenCCM\ORBacus-
4.1\OpenCCM_CONFIG_DIR\ComponentServers\ComponentServer1.archive_cache
directory.
Launching an OpenCCM's Java Component Server.
The OpenCCM's Java Component Server ComponentServer1 is started.

C:\OpenCCM\demo\demo3>jcs_start ComponentServer2
The OpenCCM's Java Component Server ComponentServer2 will be started.
Creating the C:\OpenCCM\ORBacus-
4.1\OpenCCM_CONFIG_DIR\ComponentServers\ComponentServer2.archive_cache
directory.
Launching an OpenCCM's Java Component Server.
The OpenCCM's Java Component Server ComponentServer2 is started.
```

**Then add archive demo3.jar to your classpath, ie :**

```
C:\OpenCCM\demo\demo3>set CLASSPATH=archives\%1.jar;%CLASSPATH%
```

**Getting the IOR of the started Name Service NSIOR**

```
C:\OpenCCM\demo\demo3>ns_ior
IOR:000000000000002a49444c3a6f6f632e636f6d2f436f734e616d696e672f4f424e616d6
96e67436f6e746578743a312e300000000000000010000000
```

```
0000000f3133342e3230362e31302e31343500000d31000000000028abacab305f526f6f745
04f4100526f6f74436f6e74657874504f4100004e616d655
0001000000010000002c0000000000010001000000040001002000010109000101000501000
100010109000000020001010005010001
```

## 4.4 Starting a Java Virtual Machine to start demo3

```
C:\OpenCCM\demo\demo3>bin\start_java
The OpenCCM Platform will be installed.
Creating the C:\OpenCCM\ORBacus-4.1\OpenCCM_CONFIG_DIR directory.
Creating the C:\OpenCCM\ORBacus-4.1\OpenCCM_CONFIG_DIR\ComponentServers
directory.
The OpenCCM Platform is installed.
The Name Service will be started.
Launching the ORBacus Name Service.
The Name Service is started.
The OpenCCM's Java Component Server ComponentServer1 will be started.
Creating the C:\OpenCCM\ORBacus-
4.1\OpenCCM_CONFIG_DIR\ComponentServers\ComponentServer1.archive_cache
directory.
Launching an OpenCCM's Java Component Server.
The OpenCCM's Java Component Server ComponentServer1 is started.
The OpenCCM's Java Component Server ComponentServer2 will be started.
Creating the C:\OpenCCM\ORBacus-
4.1\OpenCCM_CONFIG_DIR\ComponentServers\ComponentServer2.archive_cache
directory.
Launching an OpenCCM's Java Component Server.
The OpenCCM's Java Component Server ComponentServer2 is started.
Starting demonstration demo3 ...
Initializing the ORB...
Obtaining the Name Service...
Obtaining Component Servers...
Installing archives...
Property file null\jidlscript.properties not found: exiting
Property file null\jidlscript.properties not found: exiting
Instantiating homes...
Creating components...
Configuring components...
Interconnecting components...
Configuration completion...
Demonstration demo3 is ready to be used ...
```

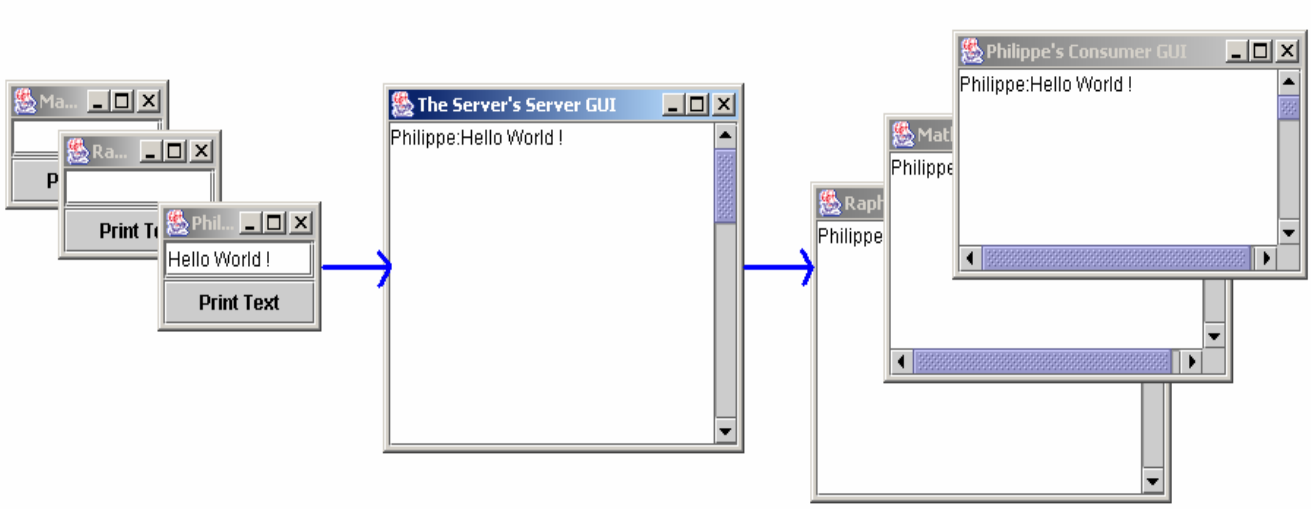



**Figure 10 – The Application at Work**

## 4.5 Stop the demo3

To stop the demo3 application, run :

```
C:\OpenCCM\demo\demo3>bin\stop_java
The Java Component Server ComponentServer1 will be stopped.
The Java Component Server ComponentServer1 is stopped.
The Java Component Server ComponentServer2 will be stopped.
The Java Component Server ComponentServer2 is stopped.
The Name Service will be stopped.
The Name Service is stopped.
The OpenCCM Platform will be deinstalled.
Removing the C:\OpenCCM\ORBacus-4.1\OpenCCM_CONFIG_DIR directory.
The OpenCCM Platform is deinstalled.
```

# 5 XML DESCRIPTORS FOR PACKAGING, ASSEMBLING AND DEPLOYING THE APPLICATION

XML descriptors help to :

- Package components into a self-descriptive package
- Assembly component with other components (if needed)
- Deploy components

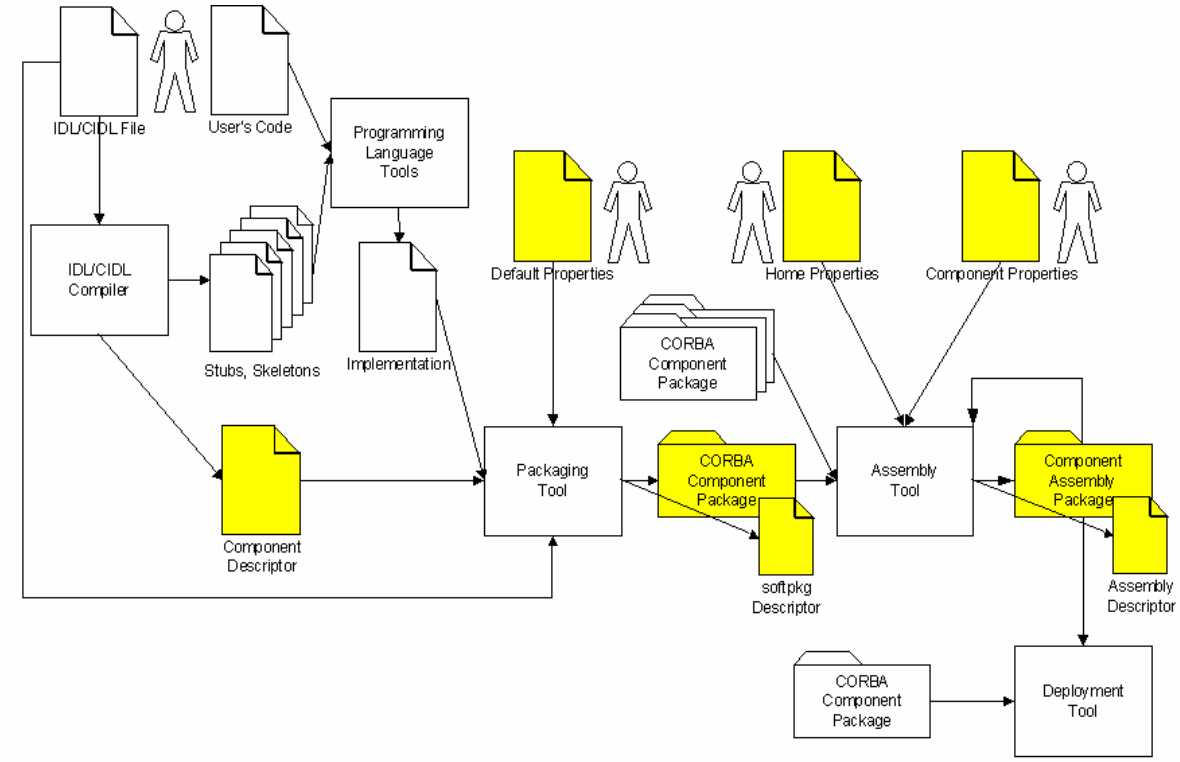



**Figure 11 – Using XML Descriptors for Packaging, Assembling and Deploying the Application**

## 5.1 Software Package Descriptor

client.csd, server.csd and consumer.csd describe general elements (title, author, description, web page, license, link to IDL file...etc) and list implementations (information about implementations like OS, ORB, language, compiler, dependancies on other libraries...etc. , entry point)

XML Software Package Descriptor for Server component (**server.csd**):

```
<?xml version="1.0"?>
<!DOCTYPE softpkg SYSTEM "../../src/dtd/ccm/softpkg.dtd">

<softpkg name="Server" version="2,0">
```

```
  <title>Server</title>
  <pkgtype>CORBA Component</pkgtype>
  <author>
    <name>...</name>
    <name>...</name>
    <company>LIFL</company>
    <webpage href="http://corbaweb.lifl.fr/OpenCCM/"/>
  </author>
  <description>Illustrating a simple client/server-producer/consumers
application</description>
  <license href="http://corbaweb.lifl.fr/OpenCCM/COPYRIGHT"/>
  <idl id="IDL:demo3/Server:1.0">
    <link href="../demo3.idl3"></link>
  </idl>
  <descriptor type="CORBA Component">
    <fileinarchive name="META-INF/server.ccd"/>
  </descriptor>

  <implementation id="ServerImpl">
    <os name="WinNT" version="4,0,0,0"/>
    <os name="Linux" version="2,2,17,0"/>
    <processor name="x86"/>
    <compiler name="JDK"/>
    <programminglanguage name="Java"/>
    <dependency type="ORB" action="assert">
      <name>OpenORB</name>
    </dependency>
    <code type="Java class">
      <fileinarchive name="archives/demo3.jar"/>

<entrypoint>org.objectweb.ccm.demo3.ServerHomeImpl.create_home</entrypoint>
    </code>
    <runtime name="Java VM" version="1,2,2,0"/>
    <runtime name="Java VM" version="1,3,0,0"/>
  </implementation>
</softpkg>
```

## 5.2 Property File Descriptor

The XML Property File Descriptor (*.cpf) is used to set home and component properties.
It contains pairs of name/value to configure home and component attributes.

Here is the property file descriptor for a client component instance : **client.cpf**

```
<?xml version="1.0"?>
<!DOCTYPE properties SYSTEM "../../src/dtd/ccm/properties.dtd">
```

```
<properties>
  <simple name="name" type="string">
    <description>Client name</description>
    <value>a guy</value>
  </simple>
</properties>
```

## 5.3 Component Assembly Descriptor

The **demo3.cad** XML Component Assembly Descriptor :

- References component software descriptors client.csd, server.csd and consumer.csd

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE componentassembly
  SYSTEM "../../src/dtd/ccm/componentassembly.dtd">
<componentassembly id="demo3">
  <componentfiles>
    <componentfile id="Client">
      <fileinarchive name="client.csd">
      </fileinarchive>
    </componentfile>
    <componentfile id="Server">
      <fileinarchive name="server.csd">
      </fileinarchive>
    </componentfile>
    <componentfile id="Consumer">
      <fileinarchive name="consumer.csd">
      </fileinarchive>
    </componentfile>
  </componentfiles>
```

- Defines home instances and their collocation, component instances
- Defines that homes, components or ports are to be registered in the ComponentHomeFinder or Naming Service

```
<partitioning>
    <homeplacement cardinality="1" id="ServerHome">
      <componentfileref idref="Server"/>
      <componentimplref idref="ServerImpl"/>
      <registerwithhomefinder name="OpenCCM/ServerHome"/>
      <registerwithnaming name="OpenCCM/ServerHome"/>
      <componentinstantiation id="Server">
        <componentproperties>
          <fileinarchive name="server.cpf">
```

```
          </fileinarchive>
        </componentproperties>
      </componentinstantiation>
      <destination>ComponentServer2</destination>
    </homeplacement>
    <hostcollocation>
    <homeplacement cardinality="1" id="ClientHome">
      <componentfileref idref="Client"/>
      <componentimplref idref="ClientImpl"/>
      <registerwithhomefinder name="OpenCCM/ClientHome"/>
      <registerwithnaming name="OpenCCM/ClientHome"/>
      <componentinstantiation id="Client 1">
        <componentproperties>
          <fileinarchive name="mathieu.cpf">
          </fileinarchive>
        </componentproperties>
      </componentinstantiation>
........
     </hostcollocation>
  </partitioning>
```

- Defines connections to be made between Client, Server and Consumer component ports (receptacles to facets, event sinks to event sources)

```
<connections>
    <connectinterface>
      <usesport>
        <usesidentifier>the_service</usesidentifier>
        <componentinstantiationref idref="Client 1"/>
      </usesport>
      <providesport>
        <providesidentifier>the_service</providesidentifier>
        <componentinstantiationref idref="Server"/>
      </providesport>
    </connectinterface>
    <connectinterface>
      <usesport>
        <usesidentifier>the_service</usesidentifier>
        <componentinstantiationref idref="Client 2"/>
      </usesport>
      <providesport>
        <providesidentifier>the_service</providesidentifier>
        <componentinstantiationref idref="Server"/>
      </providesport>
    </connectinterface>
    <connectinterface>
      <usesport>
        <usesidentifier>the_service</usesidentifier>
```

```
        <componentinstantiationref idref="Client 3"/>
      </usesport>
      <providesport>
        <providesidentifier>the_service</providesidentifier>
        <componentinstantiationref idref="Server"/>
      </providesport>
    </connectinterface>
    <connectevent>
      <consumesport>
        <consumesidentifier>from_servers</consumesidentifier>
        <componentinstantiationref idref="Consumer 1"/>
      </consumesport>
      <publishesport>
        <publishesidentifier>to_consumers</publishesidentifier>
        <componentinstantiationref idref="Server"/>
      </publishesport>
    </connectevent>
    <connectevent>
      <consumesport>
        <consumesidentifier>from_servers</consumesidentifier>
        <componentinstantiationref idref="Consumer 2"/>
      </consumesport>
      <publishesport>
        <publishesidentifier>to_consumers</publishesidentifier>
        <componentinstantiationref idref="Server"/>
      </publishesport>
    </connectevent>
    <connectevent>
      <consumesport>
        <consumesidentifier>from_servers</consumesidentifier>
        <componentinstantiationref idref="Consumer 3"/>
      </consumesport>
      <publishesport>
        <publishesidentifier>to_consumers</publishesidentifier>
        <componentinstantiationref idref="Server"/>
      </publishesport>
    </connectevent>
  </connections>
```