# A DSP MICRO-FRAMEWORK (DSP µF) FOR OMG SWRADIO SPECIFICATION EXTENSION

Frederic Lafaye, Eric Nicollet
Thales Communications, 160 boulevard de Valmy, BP 82,92704 Colombes Cedex, France
e-Mail: { frederic.lafaye,eric.nicollet}@fr.thalesgroup.com
Phone: +33(0) 1 46{132 481, 132 132}; Fax: +33(0)1 46 132 555)
.

## ABSTRACT

The paper begin with the impact of Software Defined Radio on DSP Operating Environment with a focus on the technical services.

It aims at describing the specification of the DSP µF. A focus is made on the dynamic loading and the connection process.

It will also describe its implementation in the Flexible Base Station Demonstration works made during the E2R European project.

The paper then gives metrics concerning the footprint of the solution in comparison with others related solutions.

## 1. INTRODUCTION

Most of today radio equipments try to offer more flexibility in terms of deploying several Waveforms (a.k.a RAT: radio access technologies) on a given set of hardware base band boards .The OMG Software Radio Spec[3] or The SCA[2] gives a reference unified approach for every Waveform no matter their components distribution. Nevertheless, in processors not supporting CORBA, these standards have to face low footprint and real-time constraints requirement. Following common usage, processors not supporting CORBA are denoted DSP in this paper whatever are their nature (General Purpose or Signal Processing one).

The DSP µF we describe in compliant with OMG Software Radio Spec requirements. Those are the resource deployment through the Executable Device Interface, the access to the resource through the Resource interface and last but not least the connection of Ports via any connectivity technology. This connection capability also applies to any Device Port available on the DSP.

## 2. WAVEFORM DEPLOYMENT IMPACT

### 2.1. DSP Operating Environment

A Waveform Implementation is deployed and configured by a Core Framework that take the information from an XML descriptors database.

The Core Framework and the Assembly Controller use a set of operations specified by the OMG Software Radio Spec in order to create a Resource in the Processing Unit, make the assembly with the other components constituting the Waveform Implementation and configure it.

Those Operations must be supported by the Operating Environment of the Processing Unit which is the resident software of the Platform.

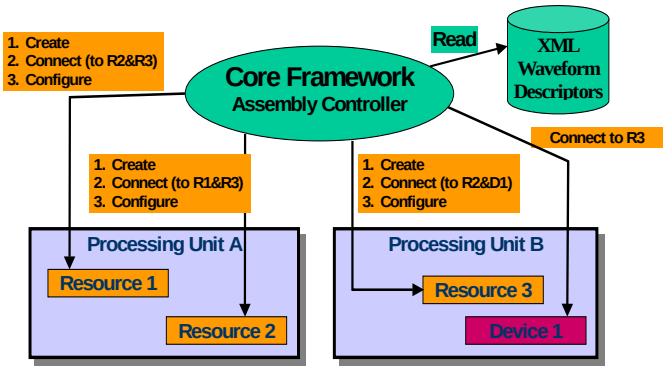


**Figure 1 : Waveform Deployment by a Core Framework**

### 2.2. Ports

Ports are Implementation artifacts supporting the connection needs. They depends on the available Connectivity Mechanisms. CORBA is the reference Connectivity to realize ports connections thanks to IDL description of interfaces between the component.

The formal description of ports and the code generation it can drive is one of the principles we keep in our approach.

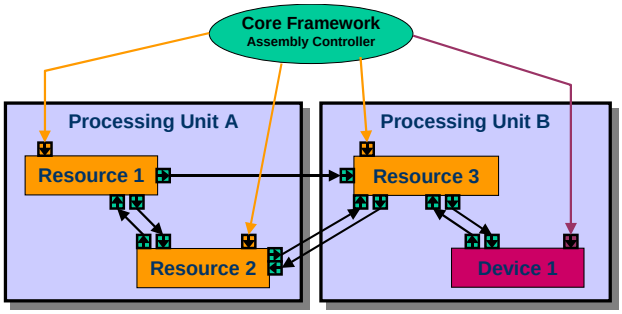


**Figure 2 : Ports**

Ports can support functional interface between Resources or between a Resource an a Device.

Ports also support management interface. Two of them are mandatory for access by a Core Framework : the Resource interface and the Device interface.

### 2.3 Resource component

A component that realize the Resource interface must support the LifeCycle, PropertySet, PortSupplier and PortConnector features:
The ControllableComponent and TestableObject features are not considered in our scope.
Those features lead to structure a Resource into :

- The Golden Source which is Independent from the Platform. That means no interaction with outside code is permitted.
- The Ports that enable the Access To/from the Golden Source through the Technical Services provided in the Processing Unit.

### 2.4. Device component

For a Device component only the PortSupplier and PortConnector features are mandatory. A Port code is to be defined for each Device component. The difference between a Device component and a Resource component is that the Business Code of the Device is tied to the Platform Hardware and for that reason is not a Golden Source which can be easily ported.
The Device component can also include some Waveform specific code. The goal is evidently to reduce them for the sake of the Waveform Application portability.
A Device has capacities that can be allocated and a State Management Feature.

### 2.5. Executable Device interface

The Executable Device is the interface that enable the creation of Resource component inside a given Processing Unit through load and execute method usage. The DSP µF implements this interface because it is mandatory in SCA to communicate with the Core Framework.

## 3. FOCUS ON TECHNICAL SERVICES DESCRIPTION

The Technical Services are defined as the Hardware and Software part of the Processing Unit that enables Software component to communicate with each other and with Hardware.

The functionality enabling that are

- the Board Support Package (BSP) for access to the hardware
- the Operating System (OS) for the Scheduling features
- the Inner Connectivity for communication between component inside the Processing Unit
- the Cross Connectivity for communication with component outside the Processing Unit.

If CORBA is a requirement of the Processing Unit, the Inner and Cross Connectivity must be realized by an ORB. If not, every solution even the most simple direct call could be envisaged.

The Device components are the part of the Technical Services that provided a “Port” Interface to the Resources.

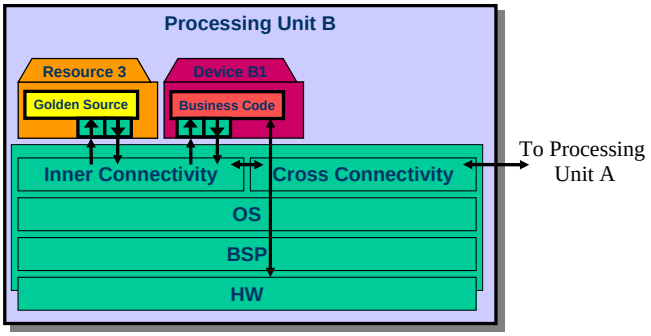


**Figure 3 : Interaction between Resource, Device and Technical Services**

One can note in the previous figure that the Business Code of the Device can have interaction with all the layers of the Technical Services whereas the Golden Source of the Resource can only communicate through the Ports of the resource he belongs to.

In other words, The separation of concerns is materialized in a Waveform that consists only of Resource Component and the Platform that consists of the Connectivity, OS, BSP and HW layers on which we add the Device Components.

## 4. EMBEDDED EXECUTABLE DEVICE SUB-SYSTEM

### 4.1. Definition

The "Embedded Executable Device" is the sub-system responsible for the Installation and Configuration of Resource components on a non-CORBA processor. It is also responsible for connecting those Resource components to other Components that can be :

- Resources residing in the same sub-system.
- Device Façade in the same sub-system.
- Resource outside the sub-system accessible with CORBA.

This sub-system is distributed on several processor:
The DSP Executable Device (shorten in ExeDevice) give the access to the DSP execution environment

The DSP µF is the part of the sub-system residing in the non-CORBA processor.

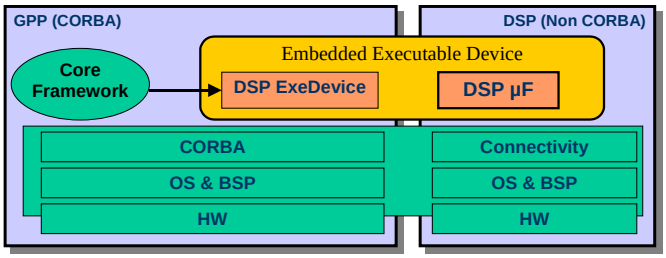


**Figure 4 : Embbeded Executable Device distribution**

Note that each component of the "Embedded Executable Device" relies on "Technical Services" described in section 3.

On the GPP, the software belonging to the "Embedded Executable Device" sub-system are based on Technical Services including CORBA

The DSP µF belongs to the Operating Environment of the non-CORBA processor. It is based on Technical Services with no CORBA requirement.

### 4.2. Relation with OMG Software Radio Spec

The following diagram show the involvement of the Embedded Executable Device in realizing the Waveform Deployment as described in paragraph 1 and implementing the OMG Software Radio Spec interfaces.

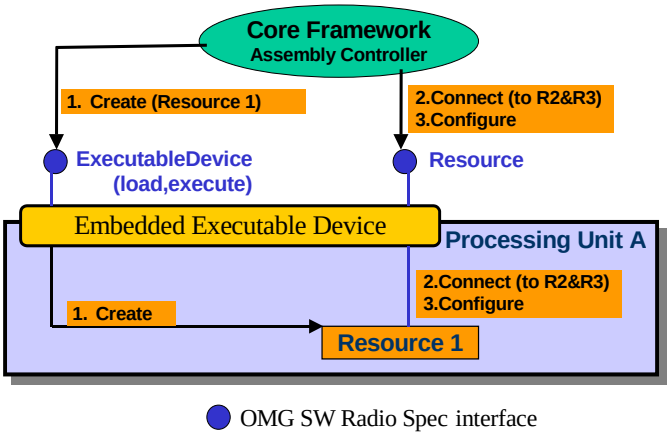


**Figure 5 : Realization of Resource Deployment in compliance with OMG SW Radio Spec**

## 5. DSP µF SPECIFICATION

The following diagram depicts the interfaces of the DSP µF with its actors.

The GPP_Resource is a component which realizes the SCA Resource interface on a Processor supporting CORBA.

The Cpp_ResourceComponent is a component that run on a non-CORBA processor. It realize ICpp_Resource wich is an Embedded implementation of the SCA Resource interface

The functional PIM interface toto between them is implemented by the ICpp_toto using the DSP µF.

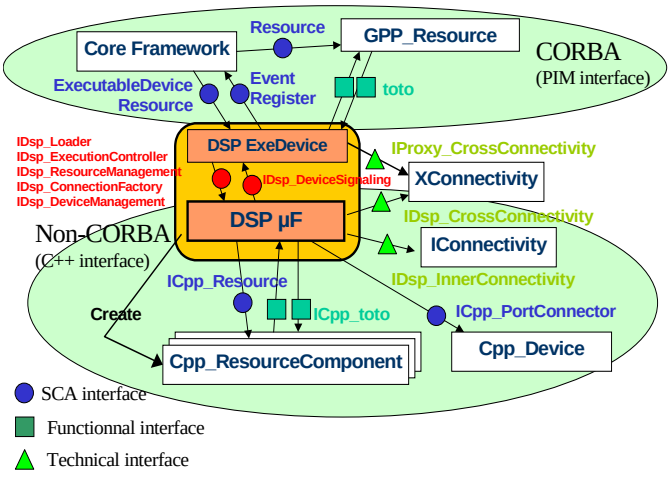


**Figure 6 : DSP µF interfaces**

Note that in CORBA space , the diagram show the Interface with the PIM name whereas in the Non-CORBA space, the interface are the C++ implementation of the OMG Software Radio Spec interface.

### 5.1. DSP ExeDevice

The ExeDevice is the proxy of the DSP_µF for the CoreFramework. It translate CoreFramework Interface to the following on the given CrossConnectivity.

The OMG SWR Spec Load/Unload is translate into the IDsp_Loader interface thanks to the BOFF format.

The OMG SWR Spec Execute/Terminate is translate into the IDsp_ExecutionController interface in order to create a Cpp_ResourceComponent that has been loaded and Terminate its execution

The OMG SWR Spec Initialize/ReleaseObject, Configure/Query, Start/Stop are translate into the IDsp_ResourceManagement.

The OMG SWR Spec GetProvidedPorts, and Connect/Disconnect are translate into the IDsp_ConnectionFactory.

The OMG SWR Spec AllocateCapacity/DeallocateCapacity is translate into the IDsp_DeviceManagement interface.

The IDsp_DeviceSignaling interface enable Auto-Registration and Signal State Transition to the ExeDevice.

### 5.2. Dynamic Load and Link of Resource Component.

The Core Framework uses the Load to dynamic load of new ResourceComponent in the Execution Environment. The DSP µF enable to load the Resource Component and dynamic link the Waveform code to the Platform code during the creation of the Resource Instance. a.k.a Cpp_ResourceComponent.

In order for the DSP µF to manage the execution of the Resource, the Cpp_ResourceComponent must implement an operation that return an Instance of himself and an operation that delete himself. like displayed in figure 11.

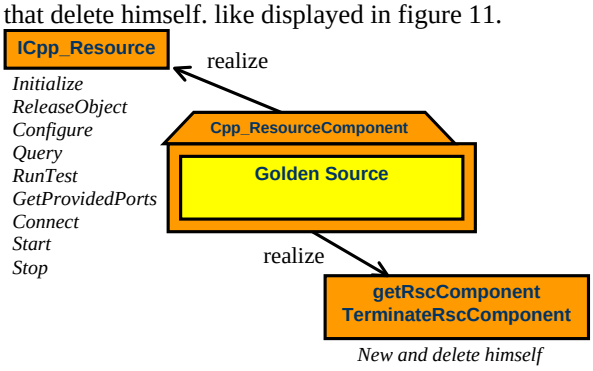


**Figure 11 : Cpp_ResourceComponent Interface requirement**

The Cpp_ResourceComponent is given by the CoreFramework as an Object File in a proprietary format called BOFF (Basic Object File Format). This file contain the object code that will be copy in the internal memory of the processing using. It also contains the raddress of the mandatory operations in order for the DSP µF to call them.

### 5.3. Establishing Connections

*5.3.1 difference between Inner and Cross connectivity*

For each Input Port, the DSP µF define an InternalChannelId which is a reference that can be used for Connection inside the same Address Space and an ExternalChannelId which is a reference passed through the CrossConnectivity.

The I_DSP_ConnectionFactory interface consist of :

- GetProvidedPorts that return for each import, the InternalChannelId and the ExternalChannelId.
- ConnectExternalPorts that enable to Connect a Resource to another residing in a different Adress Space.
- Disconnect that enable to disconnect a Resource from another (no matter their Adress Space is).

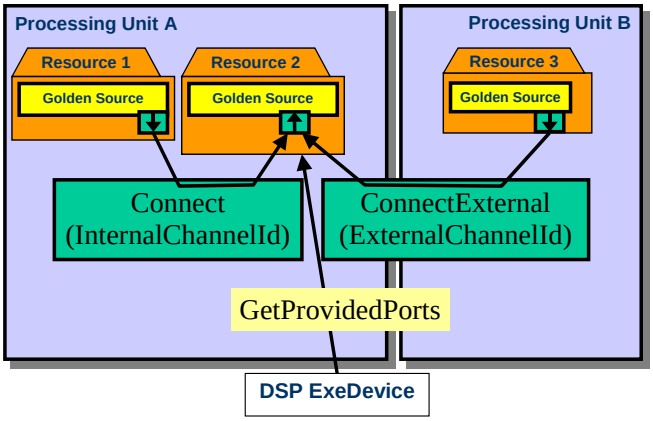


**Figure 7 : difference between Connect and ConnectExternal**

The IDsp_CrossConnectivity interface is born from the TechnicalServices requirements towards the realization of the I_DSP_ConnectionFactory .

IDsp_CrossConnectivity enable a standard access from the DSP µF point of vue by defining a Cross serializer and a Cross Initializer.

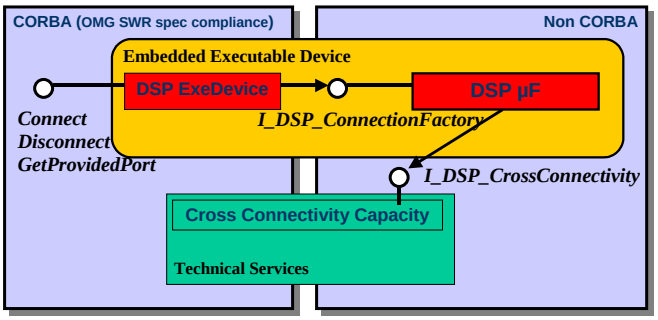


**Figure 8 : requirement towards the Technical Services**

5.3.2. Cross serializer

The cross connectivity for transmitting information could be modelized has an CrossSerializer with operation:

- SerializerRef CreateCrossSerializer (ParamCrossSerializer) for creating the mean to send data.
- DeleteCrossSerializer (SerializerRef ) for killing the Serializer created before.
- InvocateSerializer (SerializerRef, Data) for sending data through the Serializer.

On reception of a ConnectExternal, a Cross Serializer is created and connected to the Output Port of the Embedded Resource.

*5.3.3 Cross initializer*

The cross connectivity for receiving information could be modelized has an CrossInitializer with operations:

- InitalizerRef CreateCrossInitializer (ParamCrossInitializer) for Creating the mean to receive data.
- DeleteCrossInitializer(InitializerRef ) for killing the Initializer created before.
- SubscribeInitializer(InitializerRef, PortIn) for enabling the reception of Data in the InputPort.

On reception of a getProvidedPorts, for each Input Port of the embedded resource, a Cross Initializer is created and the Input Port is subscribed on it.

*5.3.4 Disconnect*

On reception of a Disconnect, the DSP_ConnectionFactory capacity enable to Deleted the Serializer if the connection was established with a ConnectExternal.
On reception of a ReleaseObject, all the Initializer Created during GetProvidedPorts are Deleted.

## 6. E2R FLEXIBLE BASE STATION DEMO

### 6.1. Implementation description

The DSP µF has been developed and connected to a Application on a PC for installing resource component.
In this Proof of concept , The interface between the PC application and the DSP µF embedded on the boards is the one specified in OMG Software Radio Spec. which is very much like the SCA.
In the context of a Flexible Base Station Demonstration in the scope of E2R European project [1], the Cross Connectivity rely on UDP sockets. The inner Connectivity is a proprietary one call BLOOD. The Processing Units are a DSP TI C5510 on a evaluation board and a ARM926 on a ST Microelectronic Greenside1 board. The Waveform is decomposed into 6 simple resources that can be mapped on every processing unit (the PC or one of the Base Band Board). The payload is a video played at a 1 Mbyte rate.

### 6.2. Demo details

The DSP µF offers multi-resource capabilities inside DSP and Multi-Application capabilities in system that used DSP. The Proof of concept elaborated in E2R is described here.

A Multi-mode Base Station used to compile several Base band board, one for each mode and each new user. With this configuration, the Need for a large number of Waveform lead to the Need for a Large number of BB Board.

The DSP µF enable to reduce the number of Base-Band board and a efficient use of the available ones.

Furthermore, the second Waveform application can be deployed with no interference to previous running one.

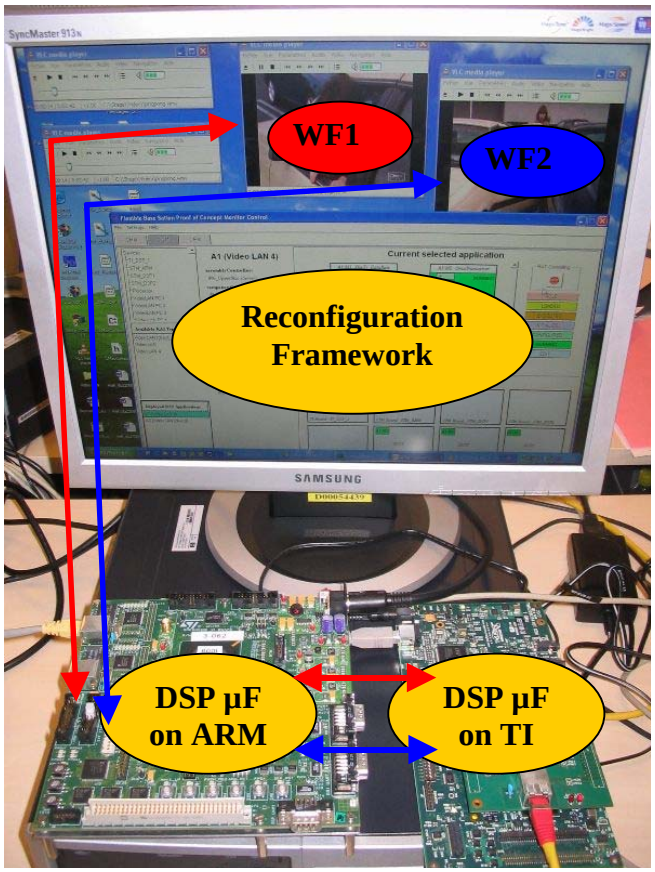


**Figure 9 : E2R FBS Demo platform**

## 7 FOOTPRINT OF THE SOLUTION

In that context, the footprint measured on the C5510x is :

| ROM | (in kBytes) | |
|---|---|---|
| 4,2 | kB | **Ressource Manager** |
| 1,5 | kB | **ExecutionController** |
| 3,2 | kB | **Loader** |
| 1,9 | kB | **ConnectionFactory** |
| 0,7 | kB | Misc |
| **11,5** | | **TOTAL** |

**Table 1 : DSP µF connectivity independent footprint**

Those data must be correlated to the footprint of the connectivity needed for the µFramework

| ROM | (in kBytes) | |
|---|---|---|
| 2,3 | kB | In Port management |
| 0,4 | kB | Out Port management |
| 9,5 | kB | µF commands |
| 3,7 | kB | µF commands init |
| **15,9** | | **TOTAL** |

**Table 2 : DSP µF connectivity dependant footprint**

Additionally to the ROM data, one can say that the RAM needed is less than 1 Kbytes

Those data must be put in front of the LightWeight CCM approach [5] which have a comparable goal but with CORBA requirement.

| | **E2R II** | mesured | **lwCCM** | typical |
|---|---|---|---|---|
| **Management** | **DSP µF** | **28 kB** | µCCM | 12 kB |
| **Connectivity** | BLOOD+UDP | 60 kB | ORB | 85 kB |
| **Scheduling** | µC-OS-II | 10 kB | OSEck | 15 kB |
| **OE** | TOTAL | **98 kB** | | **112 kB** |

**Table 2 : DSP µF solution compared to LightWeight CCM**

The UDP stack is not the optimum solution when the purpose is the footprint evaluation. This remark also apply to the lwCCM approach for which the ORB rely on TCP/IP stack.

## 8. CONCLUSION

The DSP µF we described in this paper enable to extend the very powerful concepts of the SDR inside components for which the connectivity mechanism is usually the bottleneck.

In very constrained environment such as the one we consider in DSP, the efficient Inner connectivity must enable a direct call and the efficient Cross Connectivity is usually easier than UDP. In consequence the operating environment supporting Resource management can have footprint far reduced compared to the one presented in this paper.

## 9. ACKNOWLEDGEMENT

This work was performed in project E2R II which has received research funding from the Community's Sixth Framework programme. This paper reflects only the authors' views and the Community is not liable for any use that may be made of the information contained therein. The contributions of colleagues from E2R II consortium are hereby acknowledged.

## 10. REFERENCES

[1] IST-2005-027714 E$^2$R II Project, http://www.e2r.motlabs.com/.

[2] Software Communication Architecture MSRC-5000SCA V2.x.

[3] OMG Sw Radio Spec, http://www.omg.org/

[4] W. König, K. Nolte, T.K. Lee, R.J. Jayabal., F. Lafaye, E. Nicollet. Reconfigurable Base Station Processing and Resource Allocation. 15$^{th}$ IST Mobile & Wireless Communications Summit: Budapest, july 2006.

[5] LightWeight CCM , http://www.omg.org/technology/documents/specialized_corba.htm