Component Based Software Engineering approach on DSP Targets
Motivations
Context
LwCCM/MyCCM
GPP - DSP unified approach (EULER)
Framework optimizations for DSP
Benchmarks
Perspectives
Conclusion
Motivations

- **DSP applications**
  - Lower MAC / PHY (algos, reconfigurations, servo-control,...)

- **Software constraints/challenges**
  - Increasing systems complexity, portability, reuse level

- **Software architecture efforts needed on DSP**
  - Separation of concern between technical and business
  - Focus on a global SDR approach
  - Enrich the HW processor approach of the SCA
    - IDL on GPP, MHAL Comm on DSP

- **Need of a CBSE tool-aided approach**
  - Experiment a THALES framework MyCCM
THALES is having a unique approach combining the EU R&T agenda with research for WF Portability
A THALES framework helping architects and developers to develop CBSE Distributed Real-Time Embedded applications

MyCCM = implementation of OMG Lightweight CCM

Components interact via ports
- Provided interfaces: facets
- Required connection points: receptacles
- Event sinks & sources

Components are described in IDL3 language

Our works in EULER leveraged MyCCM background towards SCA based DSP extensions

N.B: MyCCM does not postulate usage of CORBA
MyCCM Development Process

1. MAKE YOUR DESIGN

- Component specification in IDL3: interfaces, ports
- Structural & collaboration aspects (deployment)
- Real-Time tuning/constraints (deployment)
- SCA resources generated by CCM component assembly (option)
MAKE YOUR DESIGN

①

GENERATE

②

- Generation of containers for various connectivity choices
  - CORBA not mandatory
- Generation of implementation template for Business Code
- Generation of mirror components for Testing purpose
- Generation of SCA deployment XML descriptors
MyCCM Development Process

① MAKE YOUR DESIGN

② GENERATE

namespace hdrn

namespace mac

MACSupervisionImpl::SupervisorImpl::SupervisorImpl(MACSupervisionImpl& component)
{
    //INSERT YOUR BUSINESS CODE
}

③ DEVELOP YOUR BUSINESS CODE

④ BUILD

⑤ DEPLOY
Issues

- SCA resource interfaced to DSP through MHAL ports rather than functional ports
- Hand-made transformation of « would be » IDL to pushpackets (byte payload)
- Limitation to « oneway » interactions
Motivation: meeting EULER portability requirements
1 WiMax-like waveform ported onto 3 platforms

CORBA everywhere

Native Test Environment

Unified GPP-DSP approach
Native Test Environment

PC: Intel x86   Linux 2.6
Porting on THALES platform: approach

UNIFIED ID

SCA resource

HOW TO MINIMIZE PORTABILITY EFFORTS?

Legend

SCA container

CCM Component
- modem code
- legacy code

Legend

CF::Resource

SCA ports

In port

Out port

SCA connections

Specific connections

ISOLATES COMPONENT DEVELOPERS FROM GPP-DSP COMMUNICATION ISSUES

THALES Core Framework

PrismTech CORBA

GIOP(IIO-P-MQIOP)

MHAL Comm

Linux

µcOSII

GPP

DSP

IPBB

SCA container

GPP: PowerQuick II  Linux 2.6
DSP: TI C6414  µcOSII

HOW TO MINIMIZE PORTABILITY EFFORTS?
IDL/IDL3 component description

MyCCM generates 1 Resource container + 1 GPP proxy

Legend

SCA container

CCM Component
- modem code
- legacy code

Legend

SCA ports
- In port
- Out port

SCA connections

Specific connections

THALES Core Framework
PrismTech CORBA
GIOP(IIOP-MQIOP)
Linux
µcOSII

MyCCM
THALES Broker
MHAL Comm

GPP:
PowerQuick II
Linux 2.6

DSP:
TI C6414
µcOSII
Unified approach eases up porting

NO MANUAL CODE CORRECTION
FOR WF COMPONENTS
FROM ONE PLATFORM TO ANOTHER

Native Test Environment

GPP: Intel x86  Linux 2.6

OASIS

SDR4000

GPP: MPC8541
DSP: TI C6416
INTEGRITY
DSPBIOS

IPBB

GPP: PowerQuick II
DSP: TI C6414
µcOSII

GPP: Intel x86  Linux 2.6
Definition of a lightened IDL profile

Enrichment of MyCCM framework
- Specification of threading properties (active object)
- Support various memory allocators for interactions

Memory footprint reduction
- Structural modifications of the container architecture
  - inheritances, conditional compilation, optimized IDL/C++ generation
  - Footprint reduced by a factor of 5 from initial framework
Memory Footprint (Texas C6416 – 600Mhz - 1Mbytes memory)

- DSP Framework ~ 50Kbytes (5% on C6416)
  - MyCCM Runtime, Broker, MHAL Comm, POSIX subset, Allocators, …
- Components Containers

All sizes in kBytes

<table>
<thead>
<tr>
<th></th>
<th>Reference</th>
<th>Enriched</th>
</tr>
</thead>
<tbody>
<tr>
<td>No threading</td>
<td>1,7</td>
<td>1,8</td>
</tr>
<tr>
<td>Threading support</td>
<td>2,8</td>
<td>4,6</td>
</tr>
<tr>
<td>Resource interface</td>
<td>5,1</td>
<td>6,2</td>
</tr>
<tr>
<td>Connections with GPP</td>
<td>8,5</td>
<td>9,7</td>
</tr>
<tr>
<td>TOTAL</td>
<td>18,1</td>
<td>22,3</td>
</tr>
</tbody>
</table>

DSP internal memory: 1000 kB
BSP/RTOS: 100 kB
Framework: 50 kB
Per (complex) Resource: 20 kB
Per additional internal Component: 5 kB

~20% reduction on-going

TYPES

```c
typedef sequence<char,1024> payload;
typedef sequence<short,10> small_payload;
struct sType{ short _p1; long p2; small_payload p3; };
```

Benchmarks

```c
oneway void m1(in short, in sType, in long);
oneway void m2(in payload);
short m1(inout payload);
long m2(inout short);
```
Texas Instruments C6416 – 600Mhz

Co-located (No use of the middleware)

- For any connections within the DSP (up to inter-Resources)
  - Direct call (Client and Server in the same thread): \textbf{a few cycles}
  - Threaded Call - Active Object (Client & Server in separated threads, usage of message queues): \textbf{a few µs – RTOS driven}

Remote calls (Use the middleware solution)

- For any connection to a GPP component
  - Two ways call : 9 to 15 µs (deterministic allocation scheme)
  - One way call : 4 to 10 µs (deterministic allocation scheme)
  - Need to consider transport timings
    - ex: \textasciitilde 80µs for HPI 16bits with 1024bytes payload with Linux/Xenomai (GPP) & \textmu COSII (DSP) on THALES PF

- Depends on memory allocator used for exchanges management (configuration parameter)
- Take full advantage of Model-Driven approach with MyCCM
  - Early RT Analysis (e.g. usage of OMG MARTE)
  - Test Component generation
- Use of other CCM capabilities
  - Support of Events (with publish/subscribe service)
  - Support of additional interaction patterns (Connectors)
- Margins exploitable for further memory footprint optimizations
- Take advantage of ESSOR architecture and SCA Next evolutions
  - ESSOR IDL profile for DSP & FPGA
  - ESSOR MHAL Connectivity
  - Optional elementary interfaces in CF::Resource
- Evaluate potential of full MHAL solutions
COMPONENT BASED ARCHITECTURE

- Separation of concerns
- Relies on standards

TOOL-AIDED APPROACH

Structured Approach

Portability Reuse

WF Design

- Importance of Software architecture
- Experience in Real-Time Embedded Development

SDR Roadmap

WF Design

Roadmap

- Common approach for GPP and DSP teams
- Close to SCA Next preoccupation
- Reuse in ESSOR architecture
- Basis of THALES SDR DSP Operating Environments
Separation of concerns
  - Business vs Technical code

Shield middleware technical concerns to component developer

Encapsulate common execution requirements

Activation, port management, persistence, security, transactions, …

Communicate with middleware (stubs/skeletons), use middleware services
An instantiation of MyCCM for SDR on GPP

- Automatic generation of SCA resources and deployment descriptors

An instantiation of MyCCM for SDR on DSP

- Specialisation of the MyCCM for SDR for more constraint environments

- Fast adaptation to architecture requirements
  - Choices can be postponed:
    - CORBA or not CORBA
    - Native (simulation/host) or Target

- Fast integration, Easier portability
**REFERENCE COMPONENT**

```c
oneway void m1(in short, in sType, in long);
oneway void m2(in payload);
short m1(inout payload);
long m2(inout short);
```

**ENRICHEO COMPONENT**

```c
oneway void m1(in short, in sType, in long);
oneway void m2(in payload);
long m3(in long, in short);
oneway void m4(in short);
```

**TYPES**

```c
typedef sequence<char,1024> payload;
typedef sequence<short,10> small_payload;
struct sType{
    short _p1;
    long p2;
    small_payload p3;
};
```
### Benchmarks: Timings example

<table>
<thead>
<tr>
<th>Interaction Type</th>
<th>Same Thread</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>local (a1)</td>
<td>yes</td>
<td>30 cycles (~50ns)</td>
</tr>
<tr>
<td>local (a2)</td>
<td>yes</td>
<td>20 cycles (~33ns)</td>
</tr>
<tr>
<td>asynchronous (a1)</td>
<td>no</td>
<td>1794 cycles (~2.3µs)*</td>
</tr>
<tr>
<td>asynchronous (a2)</td>
<td>no</td>
<td>1062 cycles (~1.77µs)*</td>
</tr>
<tr>
<td>synchronous (s1)</td>
<td>no</td>
<td>2220 cycles (~3.7µs)*</td>
</tr>
<tr>
<td>synchronous (s2)</td>
<td>no</td>
<td>2240 cycles (~3.7µs)*</td>
</tr>
<tr>
<td>remote asynchronous (a1)</td>
<td>no</td>
<td>3791 cycles (~6.3µs)*</td>
</tr>
<tr>
<td>remote asynchronous (a2)</td>
<td>no</td>
<td>2697 cycles (~4.5µs)*</td>
</tr>
<tr>
<td>remote synchronous (s1)</td>
<td>no</td>
<td>7620 cycles (~12.7µs)*</td>
</tr>
<tr>
<td>remote synchronous (s2)</td>
<td>no</td>
<td>5740 cycles (~9.6µs)*</td>
</tr>
</tbody>
</table>

*a Memory Partition Allocator

(a1) oneway void pushData_ow(in payload, in sType)

(a2) oneway void doIt_ow(in long, in short)

(s1) void pushData(in payload, inout sType)

(s2) short doIt(in long, inout short)

### TYPES

typedef sequence<char,1024> payload;
typedef sequence<short,10> small_payload;
struct sType{
    short _p1;
    long p2;
    small_payload p3;
};