Component Based Software Engineering approach on DSP Targets

THALES

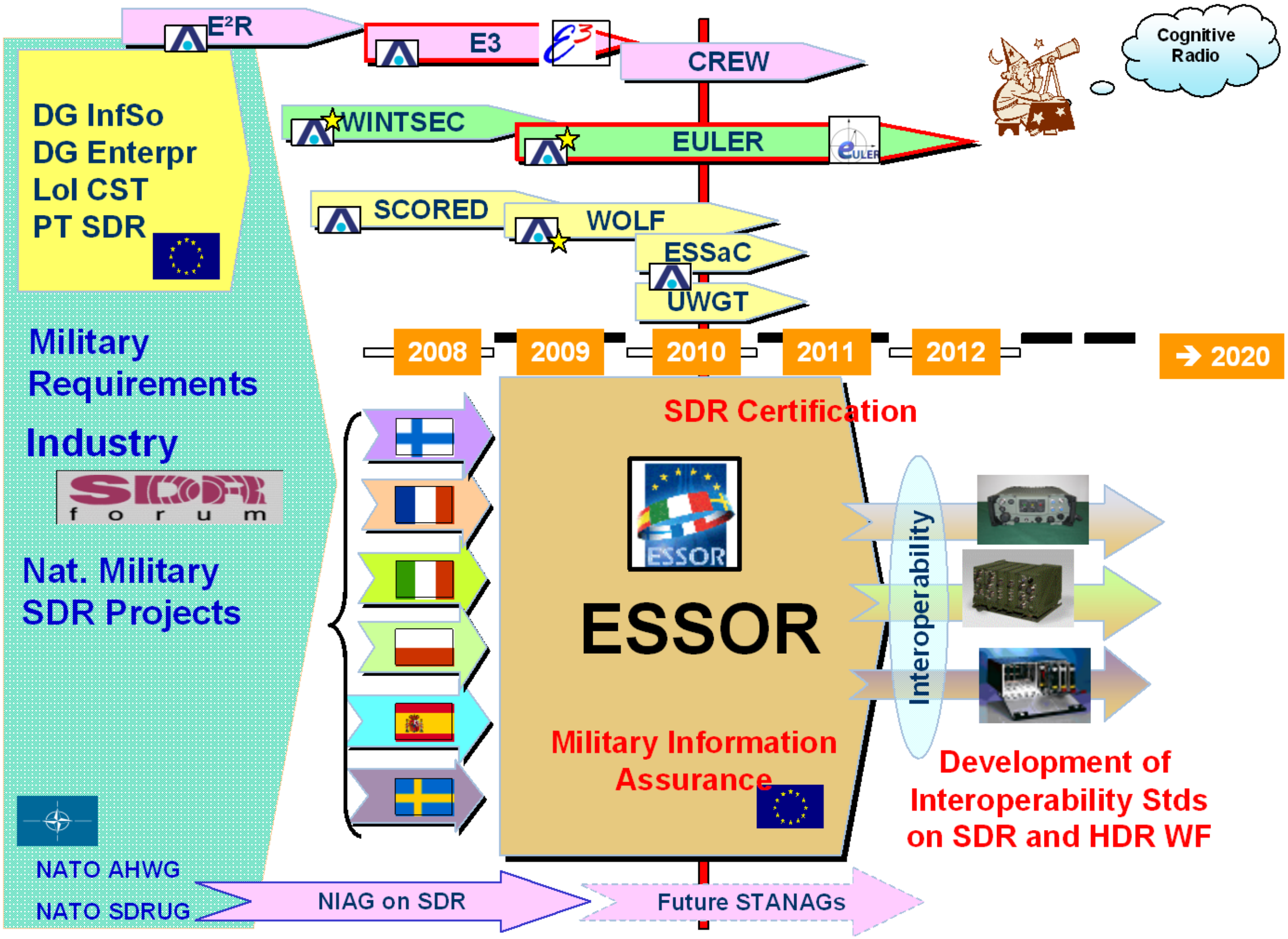
Agenda

- 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

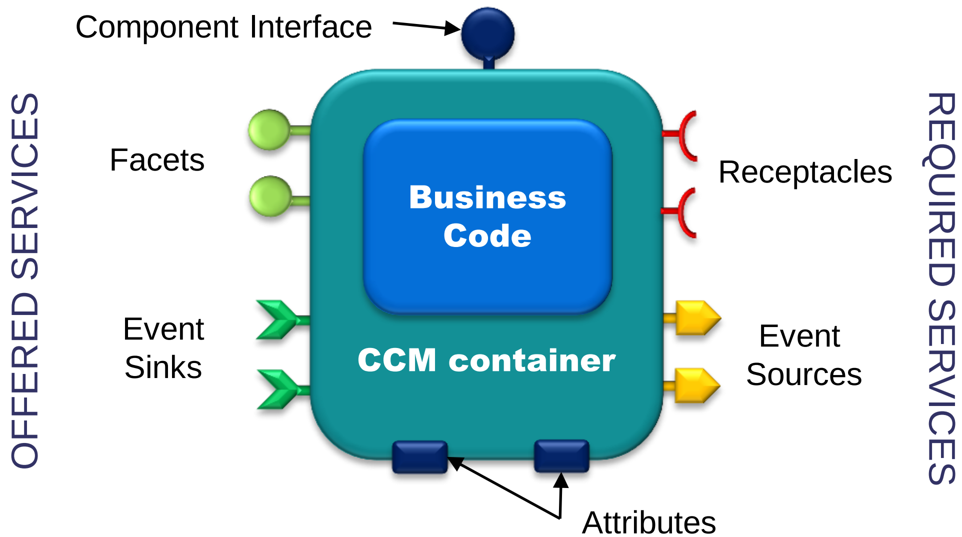
Context



THALES is having a unique approach combining the EU R&T agenda with research for WF Portability

MyCCM : MAKE YOUR COMPONENT CONTAINER MODEL

- A THALES framework helping architects and developers to develop CBSE Distributed Real-Time Embedded applications
- MyCCM = implementation of OMG Lightweight CCM



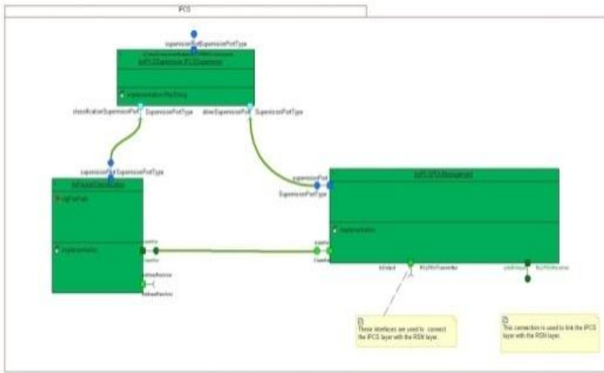
Components encapsulate application business logic

- Components interact via ports
  - Provided interfaces : facets
  - Required connection points : receptacles
  - Event sinks & sources
- Components are described in IDL3 language

N.B: MyCCM does not postulate usage of CORBA

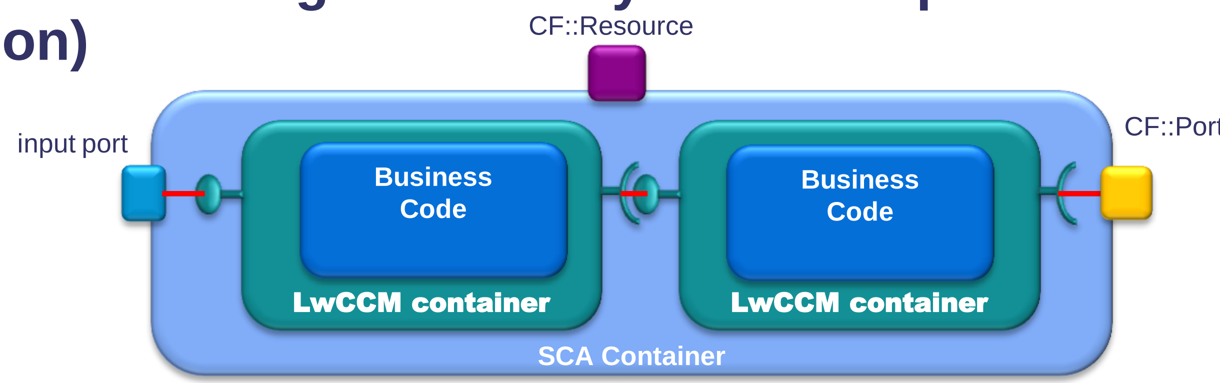
Our works in EULER leveraged MyCCM background towards SCA based DSP extensions

MyCCM Development Process

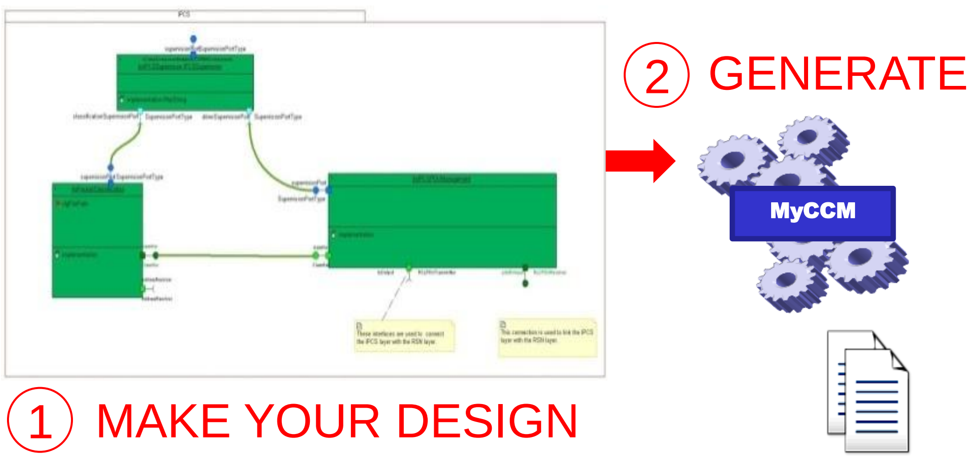


① MAKE YOUR DESIGN

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

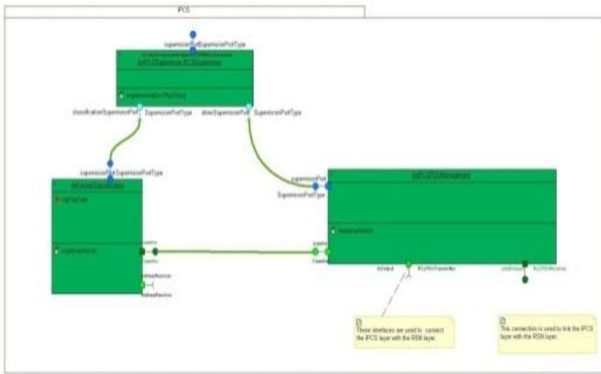


MyCCM Development Process



- 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

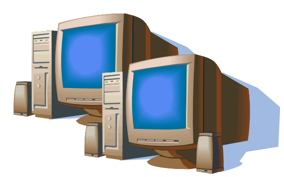
```
/*
 *  Created on: dd-mm-yyyy
 *      Author: xxxxx
 */

#include "layers/mac/MACSupervision/MACSupervisionImpl.h"
#include <assert.h>
#include <log4cxx/logger.h>

namespace hdrn
{
  namespace mac
  {
     MACSupervisionImpl::SupervisorImpl::SupervisorImpl(
                    MACSupervisionImpl& component)
     {
        //INSERT YOUR BUSINESS CODE
     }
  }
}
```

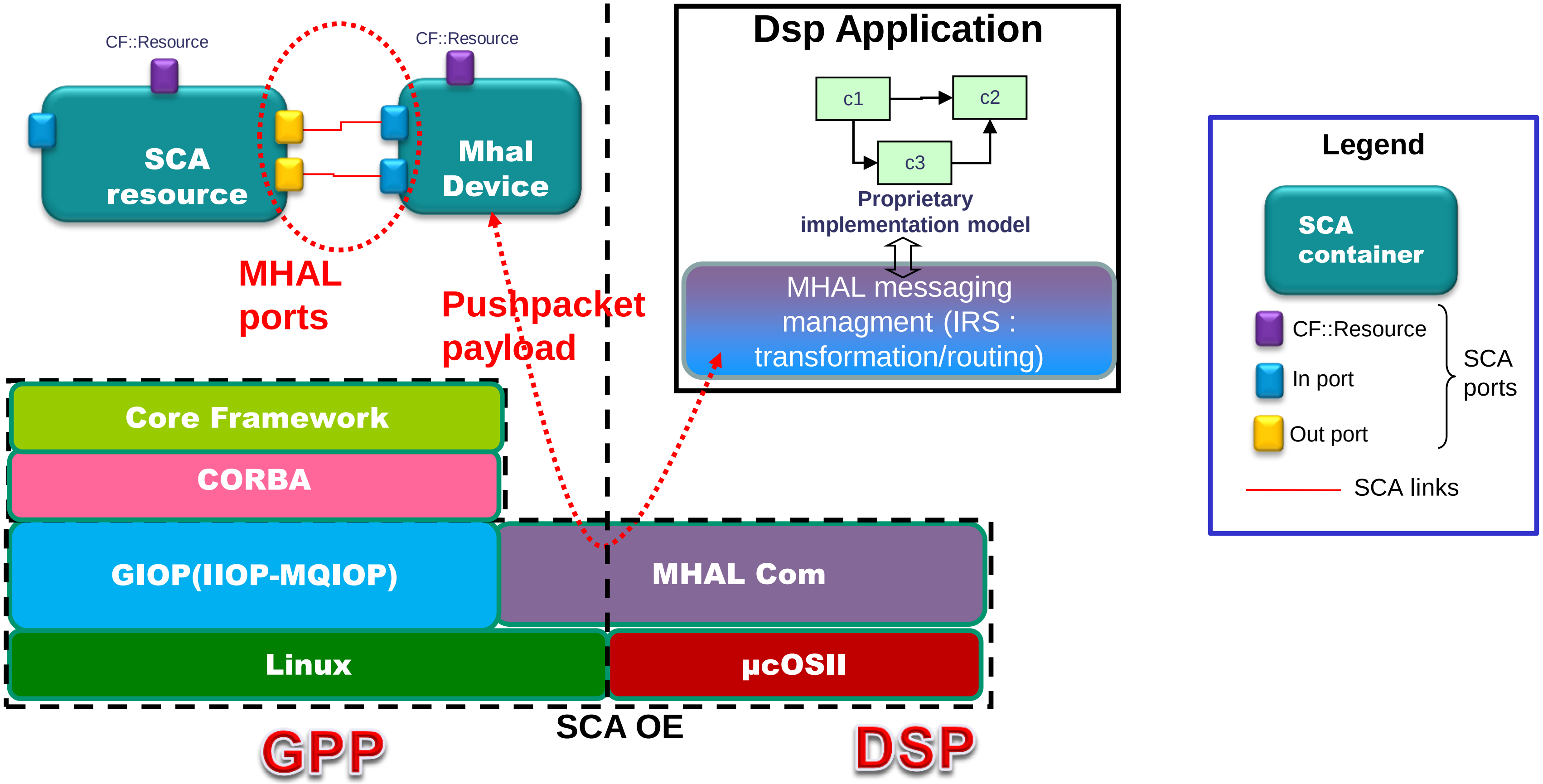
③ DEVELOP YOUR BUSINESS CODE

④ BUILD



⑤ DEPLOY

Typical SCA 2.2.2 architecture

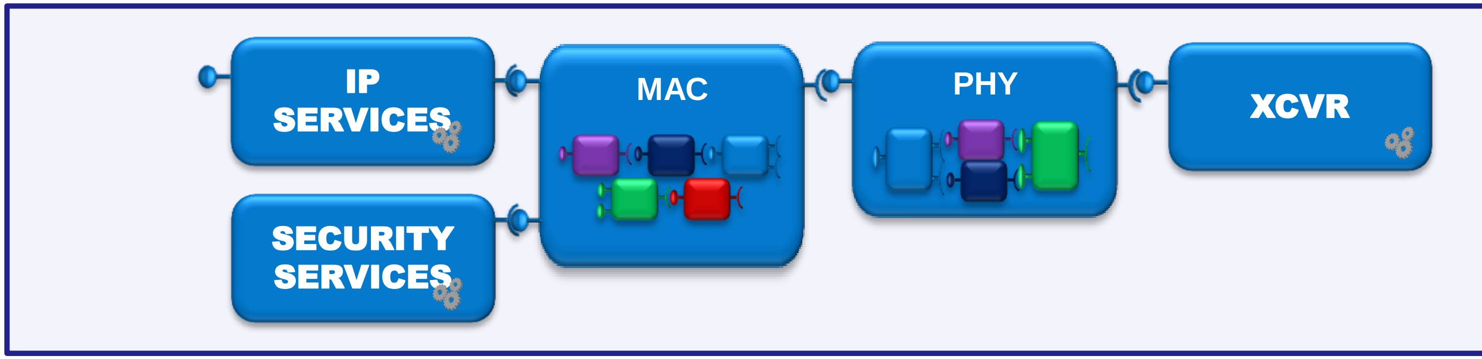


- Issues
  - SCA resource interfaced to DSP through MHAL ports rather than functional ports
  - Hand-made transformation of « would be » IDL to pushpackets (byte payload)
  - Limitation to « oneway » interactions

Unified GPP-DSP approach

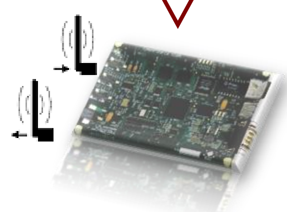
Motivation: meeting EULER portability requirements

1 WiMax-like waveform ported onto 3 platforms



GPP: Intel x86 Linux 2.6

Native Test Environment



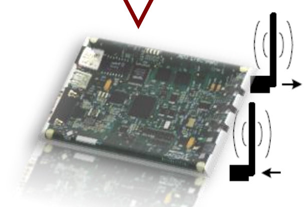
GPP: MPC8541 INTEGRITY

DSP: TI C6416 DSPBIOS

Prismtech platform

based on Spectrum SDR4000

CORBA everywhere



GPP: PowerQuick II Linux 2.6

DSP: TI C6414 µcOSII

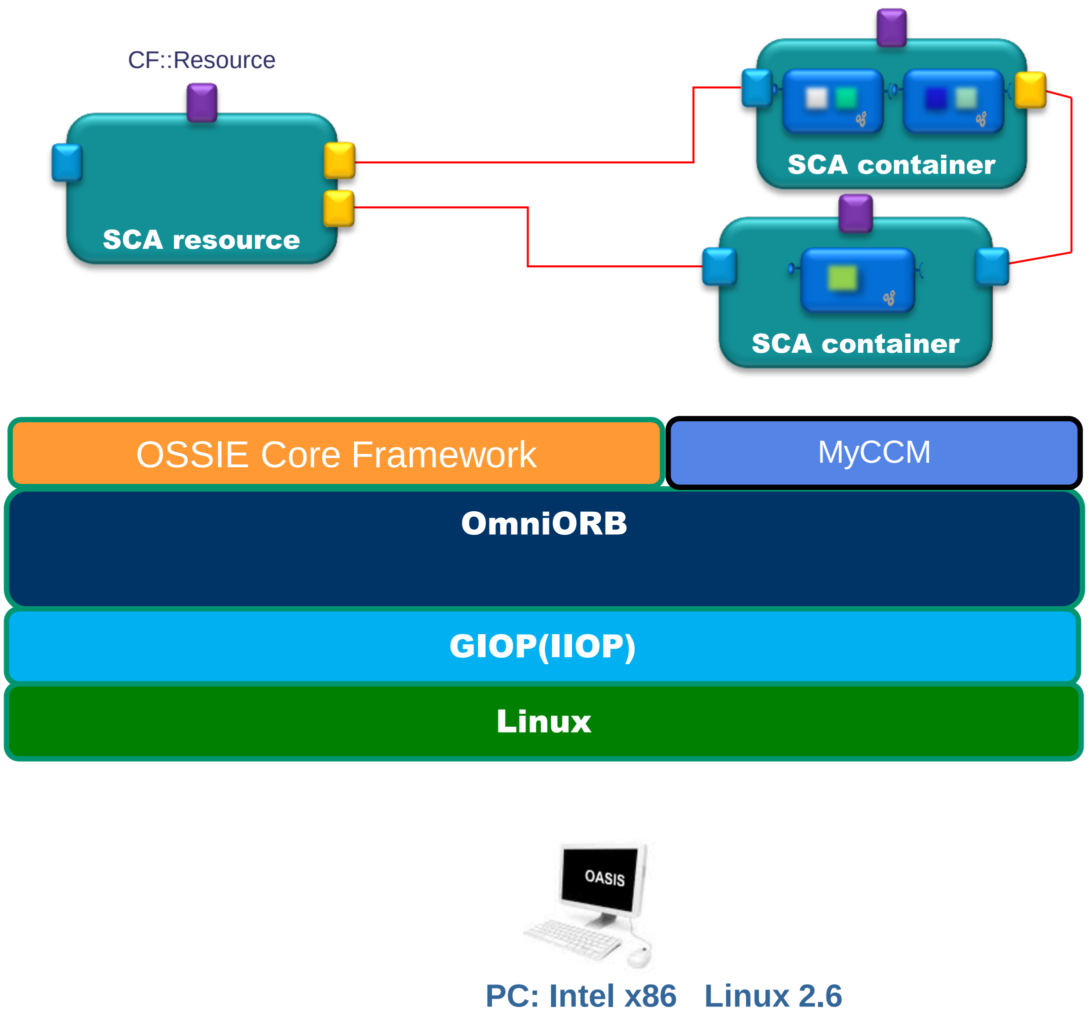
THALES platform

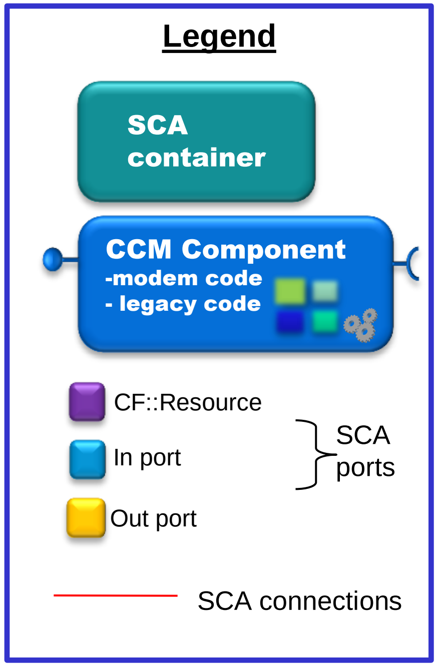
based on THALES IPBB

MHAL Comm based

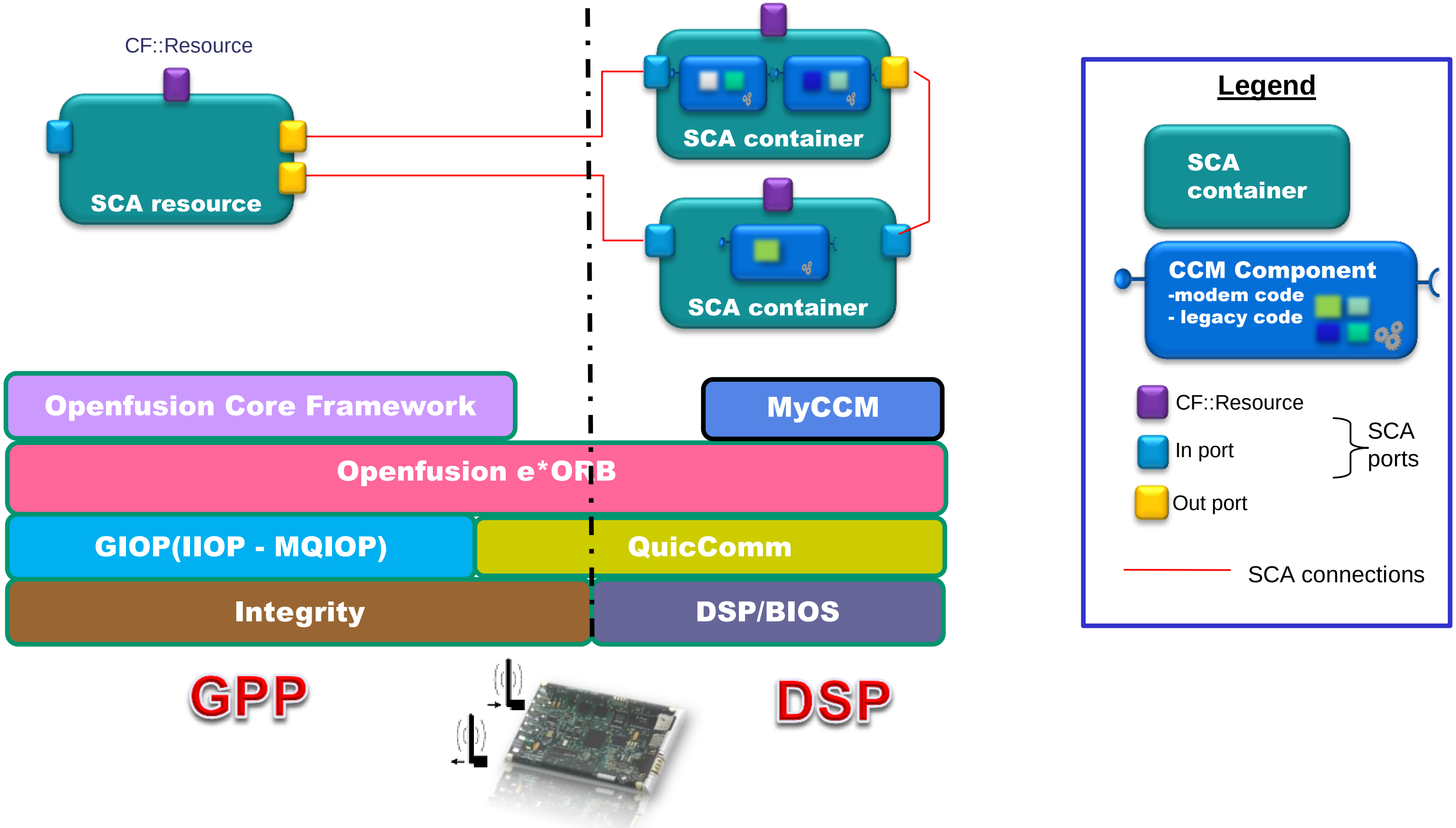
EULER THALES

Native Test Environment





Porting on Prismtech platform



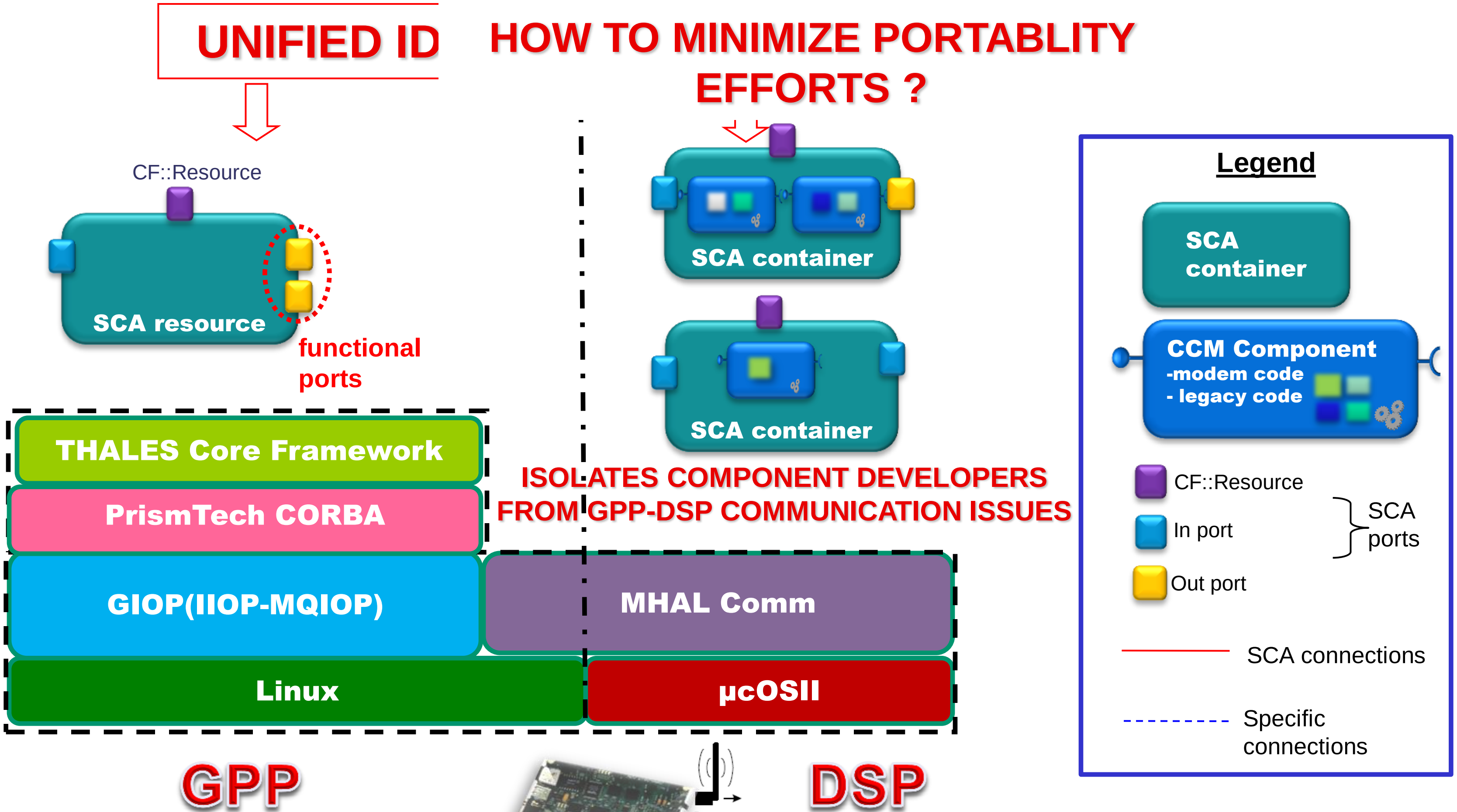
GPP: MPC8541 INTEGRITY

DSP: TI C6416 DSPBIOS

SDR4000

EULER THALES

Porting on THALES platform : approach



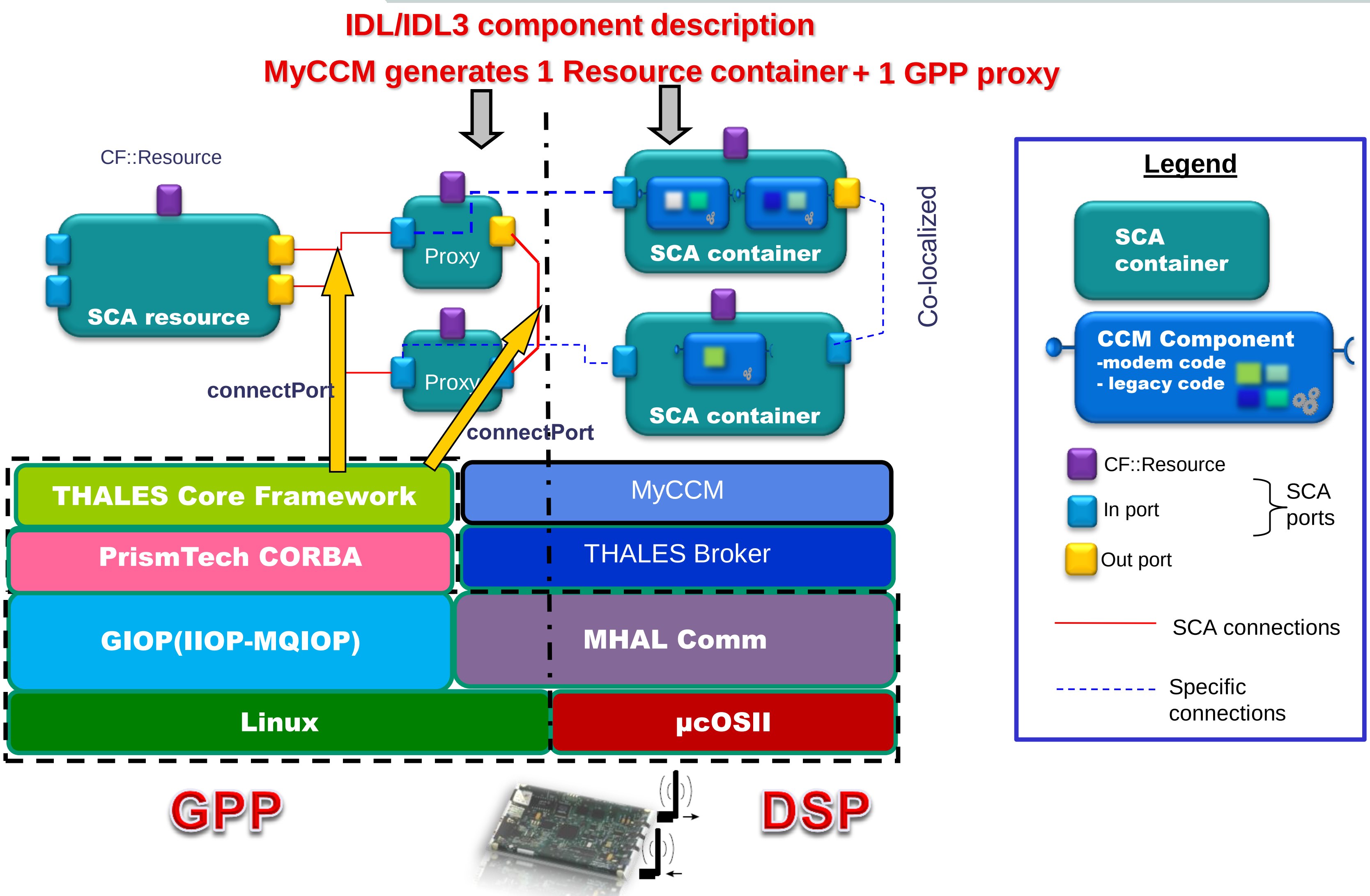
GPP: PowerQuick II Linux 2.6

DSP: TI C6414 µcOSII

IPBB

eULER THALES

Porting on THALES platform : solution



GPP: PowerQuick II Linux 2.6

DSP: TI C6414 µcOSII

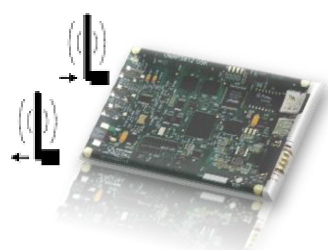
IPBB

Unified approach eases up porting

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

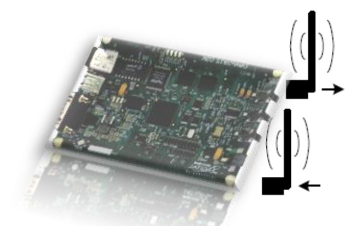
GPP: Intel x86 Linux 2.6

Native Test Environment



GPP: MPC8541 INTEGRITY
DSP: TI C6416 DSPBIOS

SDR4000

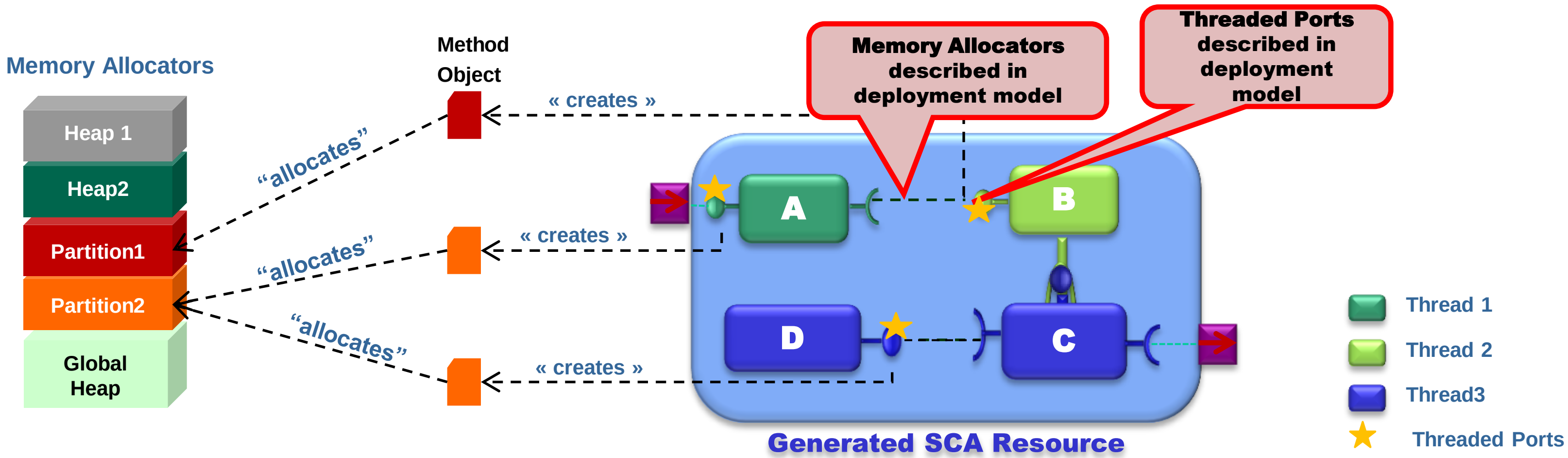


GPP: PowerQuick II Linux 2.6
DSP: TI C6414 µcOSII

IPBB

Framework Optimisations for DSP

- 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

Benchmarks

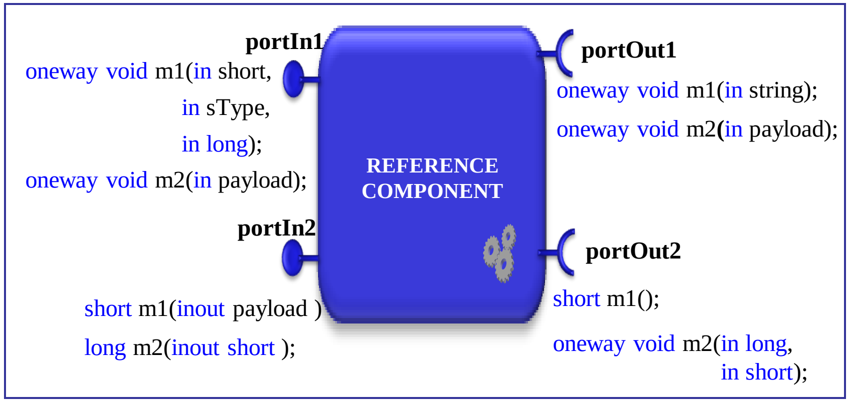
- **Memory Footprint (Texas C6416 – 600Mhz - 1Mbytes memory)**
  - **DSP Framework** ~ 50Kbytes (**5%** on C6416)
    - MyCCM Runtime, Broker, MHAL Comm, POSIX subset, Allocators, …
  - **Components Containers**

All sizes in kBytes

	Reference	Enriched
No threading	1,7	1,8
Threading support	2,8	4,6
Resource interface	5,1	6,2
Connections with GPP	8,5	9,7
TOTAL	**18,1**	**22,3**

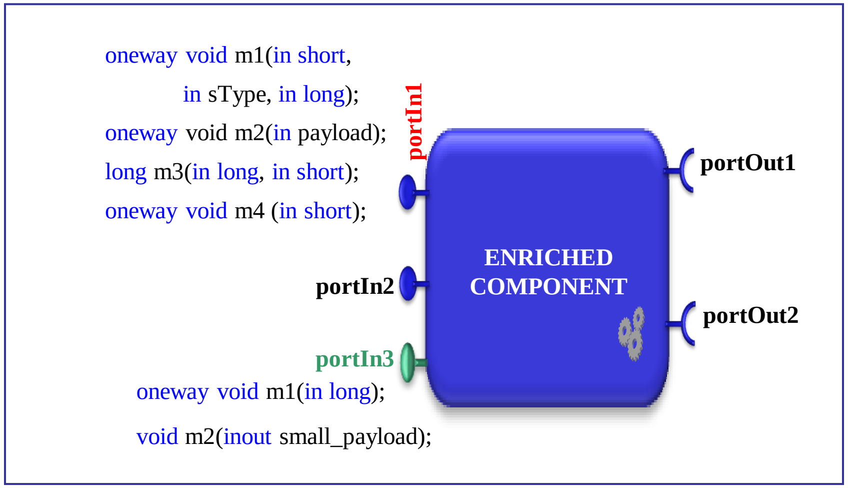
~20% reduction on-going

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



TYPES

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



Benchmarks: Communication Timings

- 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): **a few cycles**
  - **Threaded Call - Active Object** (Client & Server in separated threads, usage of message queues): **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: ~80µs for HPI 16bits with 1024bytes payload with Linux/Xenomai (GPP) & µCOSII (DSP) on THALES PF
- Depends on memory allocator used for exchanges management (configuration parameter)

Perspectives

- 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

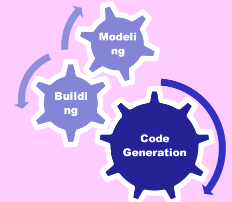
Conclusion

COMPONENT BASED ARCHITECTURE

- ✓ Separation of concerns
- ✓ Relies on standards

TOOL-AIDED APPROACH

MyCCM



Structured Approach

Portability Reuse

WF Design

SDR Roadmap

WF Design

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

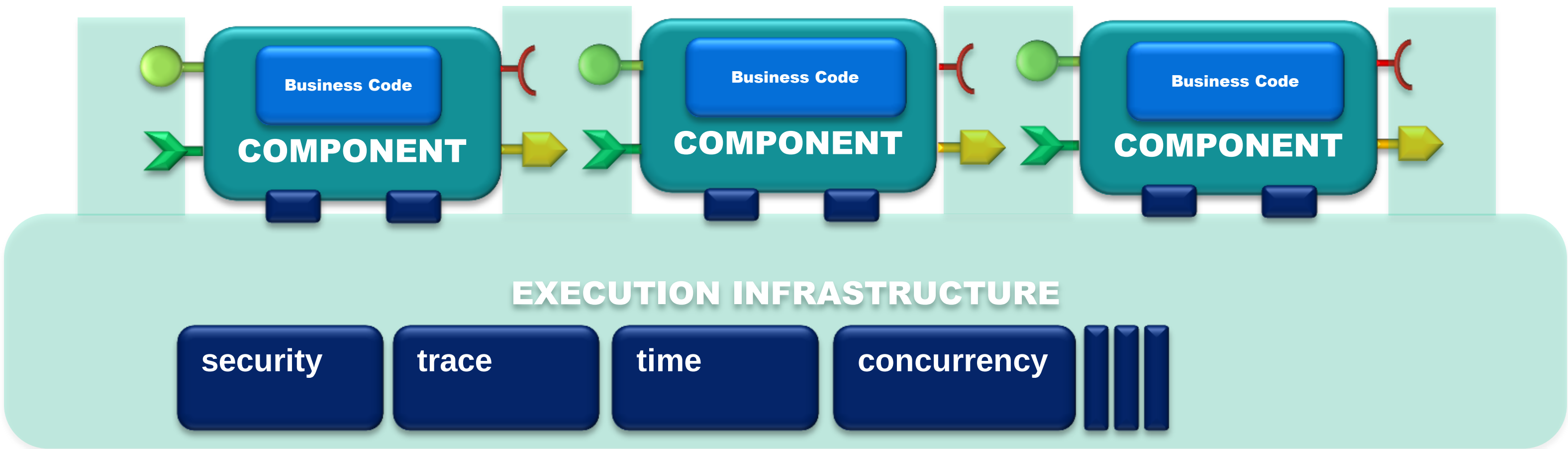
Roadmap

- ✓ Common approach for GPP and DSP teams
- ✓ Close to SCA Next preoccupation
- ✓ Reuse in ESSOR architecture
- ✓ Basis of THALES SDR DSP Operating Environments

EULER THALES

BACKUP

LwCCM Containers



- 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

Benchmarks: Components

REFERENCE COMPONENT

portIn1

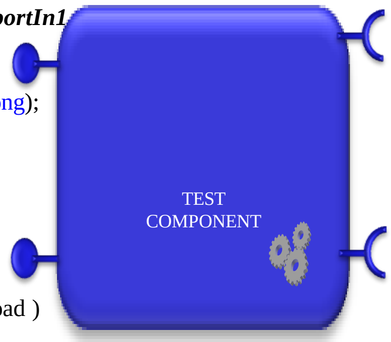
oneway void m1(in short,in sType, in long);

oneway void m2(in payload);

portIn2

short m1(inout payload)

long m2(inout short);



portOut1

oneway void m1(in string);

oneway void m2(in payload);

portOut2

short m1();

oneway void m2(in long, in short);

TYPES

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

ENRICHED COMPONENT

portIn1

oneway void m1(in short, in sType, in long);

oneway void m2(in payload);

long m3(in long, in short);

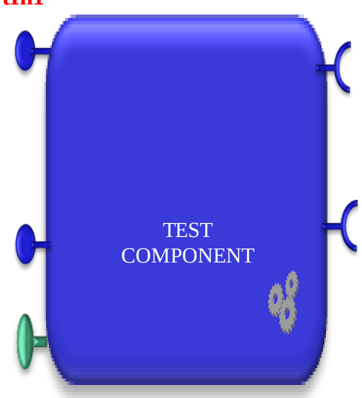
oneway void m4 (in short);

portIn2

portIn3

oneway void m1(in long);

void m2(inout small_payload);



portOut1

portOut2

Benchmarks : Timings example

Interaction Type	Same Thread	Time
local (a1)	yes	30cycles (~50ns)
local (a2)	yes	20cycles (~33ns)
asynchronous (a1)	no	1794cycles(~2,3µs)*
asynchronous (a2)	no	1062cycles(~1,77µs)*
synchronous (s1)	no	2220cycles(~3,7µs)*
synchronous (s2)	no	2240cycles(~3,7µs)*
remote asynchronous (a1)	no	3791cycles(~6,3µs)*
remote asynchronous (a2)	no	2697cycles(~4,5µs)*
remote synchronous (s1)	no	7620cycles(~12,7µs)*
remote synchronous (s2)	no	5740cycles(~9,6µs)*

* 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;
};
```