Component Frameworks for Embedded Critical Software
Industrial and Research Activities at Thales

Grégory Haïk, Thomas Vergnaud
Overview of Thales activities

Defence

50%
- Air
- Land
- Naval
- Joint

Aerospace & Space

25%

Security

25%
Who Are We? SC2 Lab @ Palaiseau, France

- 25-30 people
- Part of Land & Joint Division
  - Radio-communications, electronic warfare
  - C4I (Command, Control, Communication, Computer & Intelligence)
  - IT Security
  - Optronics, robotics, vetronics
- In charge of research & technology activities on *middleware*
  - R&T on Semantic Networks and Systems of Systems
  - R&T on component frameworks for RTE systems
    - Collaborative Research Programmes: ANR, ITEA, EU
    - THALES Group: THALES Research and Technology
    - THALES Land & Joint Systems Division
- *Transformation of R&T results* for company’s programs
Code Generation from Models

UML Models

Inter-component communication

Component Functional Code

Cont. Serv. 1
Cont. Serv. 2
Cont. Serv. 3

Technical Services (locks, trace, probes, ...)

Generation of technical code
Assisted Design and Integration

- Enforce a modular design
  - Functional chains
  - Function assemblies
  - Configuration of functions

- Model non-functional aspects
  - Input of code generation
  - Enable function re-use
  - Enable late configuration at integration time
  - Covers: user and middleware threads, priorities, period, locks, transport, deployment
Formalizing Co-Contractors Responsibilities

- Modelling and Implementation of Test Scenarios
  - Modelling of deployment diagrams for Unit Test, Integration Test, and Qualification Tests
  - Modelling of testing components
  - Handcrafted implementation of testing components

Component under test
MyCCM in Thales Programs

Programs
- Optronics domain
- Land Systems domain
- Electronic Warfare domain
- Air Defence domain

Plus internal studies
- Software-defined Radio
- Architecture of flight software
Platform and Deployment Variability

**Programming Language**
- C, C++, Java, Ada

**Operating System**
- Microsoft Windows
- Various Linux Distributions
- Linux/Xenomai
- Vxworks
- DSPBios
- Specific embedded runtime

**Hardware Architecture**
- Intel x86 32 and 64 bits
- PowerPCs
- ARM
- SPARC
- DSP

**Local vs. Distributed Deployment**
- Single or Multiple Machines/Processes/Partitions, i.e. address spaces

**Static or Dynamic Deployment**
- Static: executables are linked with component code
- Dynamic: executables are generic, components code is updatable at runtime
Interaction Variability

- Communication Semantics
  - Receive by push or pull
  - Send is blocking or non-blocking
  - …

- Quality of Service
  - Performances vs. Reliability

- Scheduling characteristics of communication threads
  - Nature: cyclic, sporadic, one-shot…?
  - Scheduling parameters: deadlines? priorities (range)?

- Interoperability with standard solutions
  - CORBA? DDS? JMS?

- Underlying Transport
Current Work on Integration with Orchestra

- System Engineering
- High-Level Software Architecture
- Detailed Software Design
- Business Component Impl.
- Deployable Software
- MyCCM Model
- MyCCM-XX
- Document Templates
- Doors - Requirements
- Doors - Test Descriptions
- Doors Automatic Testing Workbench
- Test Component Impl.
- Reqts spec
- Test Desc.
- Test Report
- Req Cov. Matrix

MyCCM Model

MyCCM-XX

Deployable Software

Document Templates

Doors - Requirements

Doors - Test Descriptions

Doors Automatic Testing Workbench

Test Component Impl.

Reqts spec

Test Desc.

Test Report

Req Cov. Matrix

ETR 2009

10
Evolution of MyCCM Internals

Front-End
- Code Generation Framework
  - Reliable Events C++
  - PolyORB C
  - TAO C++
  - JacORB Java
  - FastEvts C++
  - Local C++
  - Local C

Modelling

Documentation
- SLIM
- SRS
- PIDI
- VDD
- STD/R

Runtime Libraries
- Reliable Events C++
- PolyORB C
- TAO C++
- JacORB Java
- FastEvts C++
- Local C++
- Local C

Technical Services:
- AppTrace
- MesgLog
- Introspect
- SchedTrade
- PerfLog
- RWLock

Generic Executables:
- Component Hoster
- Deployment Tools

Build Framework
- Automatic Testing Framework

THALES
Rally standard modelling profiles
  - SysML for system engineering
  - MARTE for real time software engineering

Address Safety-critical Software
  - Improve component testing workbench
  - Schedulability analysis
  - Liveness analysis
  - Use deterministic programming constructs

Make a better usage of spare resources
  - Flexible scheduling
MyCCM High Integrity: Current Status

- Integration of C components
- Small footprint
  - 20 KB + 4 KB/C
  - Embeddable
  - Verifiable
- released under GPL
- Mode-based reconfiguration
  - Verifiable configurations
  - Verifiable transitions

February 2009 … http://myccm-hi.sourceforge.net/
Next steps toward a safety-critical framework

Expected qualities
- **Fault Containment** between distributed applications
- Guaranteed response time and throughput, **even in the presence of a faulty application**
- Programming language usage restricted to analyzable constructs
- Certifiability DO-178B, SIL...

Limitations
- Not applicable to security domain: presence of hidden channels – info leak
- Generated code has to be checked manually
Behavioral Analyses

Schedulability Analysis
- Implement RMA-based algorithms
- Build WCET of each thread by integrating those of the components
- MAST, Cheddar, UPAAL...

Liveness Analysis
- Check absence of deadlocks (and livelocks)
- Model checkers, Petri-Nets

Both analyses rely on
- Application functional architecture: assemblies
- Application non-functional architecture: threads, locks, deployment
- Behavioral abstractions

Prevention of stack overflow
- Build the call graph of the application
- Devise a conservative stack size for each thread
Certifiability

Restriction of generated code to certifiable programming patterns
- No higher-order (pointer-to-functions, objects, functions over functions…)
- Only fixed size data types
- No unbound loops
- No dynamic memory allocation

Verification of generated code
→ How to state that generated code is semantically equivalent to its model?
  - By testing
  - By testing of the generators: how?
  - By confronting the model and the code: rewriting systems?
  - By proof-carrying generators and proved generators: Coq?

Specification of requirements
→ How to state that the model is a sound refinement of the desired behavior?
  - Functional requirements: B method?
  - Non-functional requirements: model checking and schedulability analysis
Dynamic Scheduling

Dynamic scheduling Operating System

- MarteOS
- Frescor framework for negotiation between applications and OS

Reconfiguration of applications

- Easier with a component framework

Limitations

- Hard real time but not critical

Cool! I’ll use a more precise algo!

OK, back to regular algo

I need:
- 100% CPU time
- OK, here it is.

I need:
- 70% CPU time
- No more spare.

I need:
- 25% CPU time
- Extra CPU time

OK, here it is.

Negotiating OS

Negotiating OS
Deploying emerging component technology
- Since 2007, deployed in real time embedded systems

State-of-the-art research and techno development
- 2009-, targeting safety- and security-critical systems

Strong link with academics
- Telecom ParisTech, LIP6, INRIA, others...

25 engineers and researchers. Join us!
- gregory.haik@thalesgroup.com