Applying Component-based Software Engineering in On-board Software

22.10.2008
SciSys
Bristol, UK

Aleš Plšek, ales.plsek@inria.fr
Frédéric Loiret
Michal Malohlava
Lionel Seinturier
Philippe Merle
INRIA

- 8 research centers
- 1800 scientists, 1000 PhD students, 100 post-docs
- 150 joint research project-teams
- 186 million Euros budget, 20% from research contracts
- Industrial Relations
  - 790 active research contracts
  - 89 companies
INRIA Nord Europe, team ADAM

ADAM - Adaptive Distributed Applications and Middleware

• Component Oriented Programming
  • Fractal Component Model
• Model-Driven Engineering
• Service Oriented Architectures
  • WebServices
• Ubiquitous computing
  • Mobile Computing
  • Context Oriented Programming

The team
• 3 Profs, 2 Asist.Prof
• 2 Post-Doc
• 7 PhD students
• 6 R&D engineers
Me…

Past (2001-2006)

• Master Studies, DSRG, Charles University in Prague
• Model Checking of Software Components

Present – Since 2006

• 3rd year PhD Student, INRIA ADAM
• Research Interests
  – Component-Oriented Programming
  – Real-time Java Programming
Outline

Component-Based Software Engineering (CBSE)

Real-Time Java Specification (RTSJ)

Our Research
  • RTSJ for Fractal
  • Component-based RT OS
Outline

Component-Based Software Engineering (CBSE)

Real-Time Java Specification (RTSJ)

Our Research

- RTSJ for Fractal
- Component-based RT OS
Component-Oriented Programming

Component & Interfaces
- Black-box view
- Programming language agnostic

Hierarchical Component Models
- Composite & primitive components

Fractal Component Model
- Hierarchical component model
- Extension and adaptation
  - reflective components
- Lightweight
- Component Sharing
Component-Oriented Programming

```java
public class Bar extends Component implements Executable {

    private MyInterface myInterface;
    // Determined by Connector, set in architecture configuration

    public Bar() {
    }

    public void execute () {
        myInterface.baz("Hello!");
        myInterface.faz(12);
    }

    public void init() {
    }

    public void setCallableInterface(Callable anInterface) {
        if (((myInterface.requiredInterface).isInstance(anInterface)) {
            myInterfaceObject = (MyInterface) anInterface;
        }
    }
}
```

Benefits

- Separation of concerns
- Reuse
- Architectural abstractions
- And many others: adaptation, reflection, …
Component & Connectors

- Goals
  - Components are **reusable**
    - Component interaction logic separated from component functionality
      - Method calls from Bar to Foo are same regardless of connector type
  - Components are **easy to write**
    - Adapter not concerned with communication details

- Diagram:
  - Bar to Synchronous to Foo
  - Bar to Asynchronous to Foo
  - Bar to RTBoundCross to Foo

- Legend:
  - Red: Required/out port
  - Blue: Provided/in port
  - Red arrow: Direction of call
Component Container

Component Container
- **Non-functional** properties management
  - Lifecycle, Synchronization, Reconfiguration
- Interfaces
  - Business & **Control**
- Hidden from the application developer

Fractal **Membrane**
- Component-oriented container
- Reconfigurability of membranes
  - Tailorability
- Controllers & interceptors (connectors)
Component-based Development

Component Framework
- ToolChain Support
  - Glue-code generation
- Framework overhead
  - Execution infrastructure optimizations

Component System Development
- Building system from pre-existing components
- Separation of activities
  - development of components, testing, deployment, ...
- Price of CBSE
  - 5x component reuse

V development process for CBD
CBSE - Summary

Benefits
- Reuse
- Separation of Concerns
- Architectural Abstraction

Advanced Benefits
- Reflectivity, static/runtime adaptation

Framework Benefits
- Tool-chain support
- Glue-Code Generation
Outline

Component-Based Software Engineering (CBSE)

Real-Time Java Specification (RTSJ)

Our Research
- RTSJ for Fractal
- Component-based RT OS
Why Real-Time?

Real-time Programming

- A little interest in Real-time from the mainstream software engineering community
  - Deadlines, interruption handling, too low-level…

Real-Time Systems Trends

- Large-scale, heterogeneous systems
- Dynamically highly adaptable systems
- Systems composed from hard-, soft-, and non-real-time units

Many software engineering techniques can be applied in real-time domain
- Component oriented programming, Code generation, Model Driven Engineering, Formal Verification, etc.
Why Java?

Java
- Easy to use, familiar
- Popular programming language
- Libraries
- Portable across platforms
- But – non-predictable

RTSJ – Real-time Specification for Java
- Making Java predictable
Successful Stories

Shipboard computing

• US navy Zumwalt-class Destroyer
• 5 mio lines of Java code
• Red Hat Linux, RT GC the key part

Avionics

• 787 Dreamliner saves 900kgs of weight
• A380 saves a half of the processing units

Financial Information Systems
RTSJ – Making Java Deterministic

- **Real-time Threads**
  - 2 New Types of Threads
    - Realtime threads
    - NoheapRealtime threads
  - Real-Time threads
    - 28 Real-time priorities
  - NoheapRealtime threads
    - Can not be preempted by Garbage Collector
    - No heap memory access

- **Memory Management**
  - Immortal Memory
    - Objects are collected when the application terminates (live forever…)
  - Memory Scope
    - Size is fixed and pre-declared
    - Maximum size specified when scopes are created
    - Lifetime of objects in the Scope
Challenges in Real-Time Java

Advantages
- 1/9/90 Real-time Rule
- Standard Java Advantages
- hard-, soft-, and non-real-time cooperation

Complexities
- Error-prone process
- Non-intuitive rules and restrictions
- Introducing a new programming style

Software Engineering Aspect
- Ad-hoc approach
- No reuse, verification, formalization, etc.
- No adaptability, distribution support
RTSJ vs. C++

Project Golden Gate
- RTSJ on a Mars Rover

RTSJ vs. C++
- C++: memory management, ...
- RTSJ: scheduling API

The bottom line…
- **Essential** vs. **incidental** choices
- **Separation of concerns** needed
- Framework:
  - Essentials specification
  - Generation of language and platform-specific incidentals
Remedy?

Component Framework for Real-time Java
• To shield developers from the RTSJ complexities

State-of-the-Art Frameworks
• Compadres, Golden Gate, Real-Time Java Patterns…
• Component-Oriented frameworks for RTSJ

However:
• No separation of concerns
• Low level use of RTSJ concepts
• No adaptability of developed systems
Outline

Component-Based Software Engineering (CBSE)

Real-Time Java Specification (RTSJ)

Our Research
- RTSJ for Fractal
- Component-based RT OS
Our Goal

• Our Philosophy
  • RTSJ substantially *influences the architecture* of the system, therefore has to be considered *earlier then* during the implementation
  • Separation of Concerns

• Ultimate Goal: Component Framework for RTSJ
  • Alleviate the development process
  • *Isolate* RTSJ–related properties in clearly identified entities
  • Manipulate RTSJ-concerns during the development lifecycle
Real-Time Component Model - Advantages

Domain Specific Layer
- Domain Components
- Functional Components

General Purpose Layer
- Abstracting the complexities of real-time development
- Real-Time concerns at the architectural level
  - evaluate RTSJ compatibility earlier than "after the implementation"
Domain Components Application

- Different assemblies of real-time components - Adapting systems for different real-time conditions.

• Composition & Communication constraints
  • At the architectural level we reason about conformance to RTSJ
Execution Infrastructure, Membrane Architectures

Framework

- Glue-code generation
  - Execution Infrastructure Code
  - Membrane architecture generation
  - Intercepting mechanisms, connectors
  - Generated code **conforms to RTSJ**
Framework Summary

Component Framework for RTSJ
  • Benchmarks

Advantages
  • Separation of Concepts
    – Domain Components
  • Architectural Abstractions
    – RT concerns at the architectural level
  • RTSJ-related code generation
    – Membranes
    – Connectors
  • Mitigation of complexities
    – Only functional code implemented by the user

Execution Time Distribution

Memory Footprint
Outline

Component-Based Software Engineering (CBSE)

Real-Time Java Specification (RTSJ)

Our Research
  - RTSJ for Fractal
  - Component-based Real-Time OS
Real-time OS

Motivation

• OS implementations generally « highly monolithic »
  – Implemented as number of functions highly-coupled
  – Control based on many globally shared variables

Goals – Component-oriented RT OS

• Enhance the code modularity & the reuse of low-level basic services
• Tailorable OS
• Based on Think - C implementation of Fractal
• Performance & memory footprint overhead impacted by the framework is a priority to considered at OS-level
Componentization of microC OS

**Micrium - μC/OS-II**

- Provide the basic real-time services, deterministic
  - Task management (priority-based scheduling), Time and Timer management, Fixed Sized Memory Block management, IPC : Semaphores / Message mailboxes and queues
- Well established in industry
- **Certified in avionics** - by the Federal Avion Adinistration (FAA)
- Ported on many hardware platforms
Real-Time OS – Component Oriented

Delay

Task Resume/Suspend

Task Create

Time Management

Decoupling between generic and hardware-dependent components

Generic Component

Hardware-Dependent component

TimeTick_ISR

Context Switch

CPU

Decoupling between generic and hardware-dependent components

Task Management

Delay

Task Resume/Suspend

Task Create

Time Management

Decoupling between generic and hardware-dependent components

Generic Component

Hardware-Dependent component

TimeTick_ISR

Context Switch

CPU
Application Example

- Multitasking aspects are specified at the architectural level
RT OS – Industrial Project RoadMap

Short-term

- Performance and memory footprint analyses compared to the original implementation
- **Minimize the overhead**
  - Suppress the indirections added by the framework
    - (Several optimizations already implemented within the Think tool chain)
- Port the experiment to a microcontroller (32 bits / ARM based)

Mid-term

- JVM componentization
- MIND project
  - French industries interested in using CBSE toolchains at the production level
  - Implementation of Operating System and Middleware component libraries for:
    - multiprocessor System-on-Chip (MPSoc)
    - E.g. electric distribution devices
Outline

Component-Based Software Engineering (CBSE)

Real-Time Java Specification (RTSJ)

Our Research
  • RTSJ for Fractal
  • Component-based Real-Time OS

Conclusion
Conclusion

Component-Oriented Programming
- Already established in industry - Fractal
- Benefits
  - Reuse, separation of concerns, architecture abstraction
- Advanced Benefits
  - Reconfigurability – hard to achieve in RT systems

RTSJ
- Not applicable without extensive tool support

Framework for RTSJ
- CBSE substantially alleviate the development process
- Separation of RTSJ- and business-related code
- RTSJ code automatically generated
Questions?