A Component Framework for Real-time Java

12.11.2008
DSRG
Prague, Czech Republic

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

• The French National Institute for Research in Computer Science and Control
• 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
  - DeployWare
- Model-Driven Engineering
- Service Oriented Architectures
- 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
  – Model Checking
Scope of the Talk

A Component Framework for Java-based Real-time Distributed Embedded Systems

Domains

• Component-Oriented Programming
• Software Engineering
• Model Driven Engineering
• Real-time Java Programming

Another Technologies

• Adaptation
• Embedded Environments
• Distributed Programming
• Framework Design
• Code Generation
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
- 5mio lines of Java code
- Red Hat Linux, RT GC the key part

Avionics
- 787 Dreamliner saves 900kg weight
- A380 saves a half of the processing units

Financial Information Systems
- 100 milliseconds deadlines
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
- Is RT garbage collection the solution?

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
Our Goal

Our Philosophy
- RTSJ substantially *influences the architecture* of the system, therefore has to be considered *earlier than* 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

**Advantages**
- **Abstracting** the complexities of real-time development
- Real-Time concerns at the architectural level
  - evaluate **RTSJ compatibility** earlier than “after the implementation”
Domain Component

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

Memory Domains

Thread Domains

• Composition & Communication constraints
  • At the architectural level we reason about the conformance to RTSJ
Formalization of the Framework

Alloy Language
- Formalization language
- Abstract syntax, static and dynamic semantics

Goal
- Formal Verification of system’s architecture
  - Conformance to RTSJ
  - Feedback
- To allow designers to reason before starting the implementation

- Based on Fractoy
- Fractal + Alloy
Using Alloy

- Incremental Design
- Metamodel instances
- Model instances

Fractal ADL compliant
- A mapping between Alloy and ADL definition
- Formalization of Fractal ADL

RT System Architecture
- Specified and verified by Alloy
 Execution Infrastructure Development

- Execution Infrastructure Methodology
  - Functional Component Implementation
  - RTSJ Aspects
    - Using the results of the Design Process
    - Glue Code Generation
Execution Infrastructure

Key Ideas

- **Keep components** also at the implementation layer

- **Distinguish** functional and domain components
  - No bindings between functional and domain-components

- **Hidden** Intercepting and control mechanisms to manage RT concerns
Membrane Architectures

Fractal Membranes

- Interceptor and Controllers
- Domain Component
  - Specially designed membrane architecture
Preliminary Results

Performance Benchmarks

• Measuring **overhead** of the framework
• Different levels of **optimization**

**Execution Time Distribution**

**Memory Footprint**

<table>
<thead>
<tr>
<th>[KBytes]</th>
<th>100</th>
<th>200</th>
<th>300</th>
<th>400</th>
<th>500</th>
</tr>
</thead>
<tbody>
<tr>
<td>OO</td>
<td>276,8</td>
<td></td>
<td></td>
<td></td>
<td>280</td>
</tr>
<tr>
<td>Soleil</td>
<td>557,7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Merge_All</td>
<td>281,5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ultra_Merge</td>
<td>226,2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Case Study

Realtime Collision Detector

- Developed by Filip Pizlo at Purdue University

RCD Architecture
Current Status

Prototype Implementation Development

- Based on Fractal, Juliac
- **Soleil** Project
  - Execution Infrastructure Generator
- **Hulotte** Project
  - Membrane Extensions, Connectors
  - Distribution for RTSJ

Formalization of the Framework

- Alloy Analyzer
- Joint work with **Philippe** (Fractoy Project)

Case Study

- Real-time **Collision Detector System** developed at Purdue University
Framework Summary

Component Model
- Introducing a matured component model - Fractal, into the world of real-time programming
- Explicit separation of functional and real-time concerns
- Domain Components

Software Engineering
- Clarifying the system’s development lifecycle
- Formalization - System architecture in compliance with RTSJ

Execution Infrastructure
- Membrane Architectures
- Separation of Concerns
- Code Generation Approach

Prove of Concept
- Framework prototype implementation
- Benchmarks, Case Study
Future Work

Framework Implementation
  • Formalization of Membrane Architecture
    – Composition of Controllers/Interceptors

Case Study Evaluation
  • Contributions to Software Engineering of RT
  • Evaluate the ideas in a complex scenario

Activity View Generation
  • Incorporating Tinap, extracting the task model from the architecture

Support of Dynamic Adaptability
Questions?