A Component-Based QoS Architecture for Embedded Systems

Jean-Charles Tournier   France Télécom
Jean-Philippe Babau   CITI / INSA Lyon

ARTIST 2 - Component Modelling and Composition
Paris, January 24th 2005
Context - 1/2

Mobile embedded systems

Today

- Complex features
- **QoS constraints (real-time, safety, etc.)**
- Limited resources (CPU, memory, battery, etc.)
- lead to dedicated, closed and monolithic systems

Tomorrow: need for flexible systems

- **Dynamicity**
  - Add or remove applications on the fly
- **Opening**
  - Heterogeneous communication networks (GSM, GRPS, Bluetooth, etc.)
Component Based Software Engineering (CBSE)

An easier way to build complex software by assembling, composing software entities called *components*

CBSE provides

Adaptability
Reusability
Scalability

Component frameworks (Think, OS-Kit, Coyote)

Provide composition of components from a functional point of view
But, no support for QoS management
Objectives

Dynamically manage QoS constraints in component-based systems

Define a QoS architecture
  Open
    Independent of QoS management policies, Qos mechanisms and Qos specifications
  Component-based
  Supporting dynamic QoS management
Overview

Our view of components

Main QoS concepts

Qinna

Experiment

Analysis

Conclusions
CBSE - The Fractal Component Model

Component
  Run-time entity
  Made of a content and a controller

Content
  Composed of a finite number of other components (recursive)
  Under the control of the controller

Controller
  Reflexivity
  E.g. Life-cycle, configuration, etc.

Interface
  Unique access points
  Client or server

Binding
  Oriented connection between two components
    From a client interface to a server interface (type compatibility)
  Primitive or composite (component+binding)
API
Life-Cycle, Reflexivity (BindingController and ContentController), Instanciation (Factory)

ADL
A language for defining Fractal architectures (configuration)
A modular and an open (extensible) language to describe
Components, interfaces and bindings
Attributes
Typing
Implementations
Deployment
Behaviour and QoS contracts
...
CBSE - The Think Framework

Think
Framework for component-based operating systems
Conforms to the Fractal model
Everything is components
  Fine grain control over resources

Two kinds of components
Hardware Abstraction Components
  Boot, exceptions, MMU, device drivers (screen, keyboard, serial port,…), etc.
OS Services Components
  Thread, scheduler (RR, priority, EDF,…), Network (Ethernet, IP, TCP,…), etc.

Ported on ARM (Intel StrongARM, Intel xScale, Portal Player, Motorola Dragon Ball MX1),
Intel x86, Hitachi H8 (Lego RCX)

Provides a complete tools chain
  ADL and IDL parsers, component compiler, etc.
Quality of Service architecture
A Component-Based QoS Architecture
   Capturing the main QoS concepts
   Generic
   Dynamic
   Reusable

Qinna defines
   A set of APIs
       Component types
       Data types
       Inter-component relationships
       Dynamic behavior
Qinna - QoSComponent

Definition
Provides functional interfaces with QoS management needs

Detail
Qinja - QoSComponentBroker

Definition

Provides a QoSComponent with a fixed local constraint
Responsible for the admission testing and reservation for a class of QoSComponents

Detail
Qinna - QoSComponentManager

Definition

Initializes a QoSComponent from a QoS point of view
 Implements initialization, maintenance and adaptation mechanisms for a class of QoSComponents

Detail

![Diagram of Qinna - QoSComponentManager]
Qinna - QoSComponentObserver

Definition

Implements observation policy for a class of QoSComponents

Detail
Qinna - QoSDomain

Definition
Encapsulates all Qinna components
Implements adaptation and maintenance policies based on importance level

Detail

![Diagram showing QoSDomain relationship with iQoSMaintener, iQoSAdapter, iQoSManager, iQoSObserverException, and QoSObserverLifeCycle]
Experiment

Diagram:

- GUI
- Video
- DOOM
- Threads
- Memory
- EDF

Table:

<table>
<thead>
<tr>
<th>Video QoS Level</th>
<th>Thread QoS Level</th>
<th>Memory QoS Level (kb)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GOOD</td>
<td>{4,10}</td>
<td>60</td>
</tr>
<tr>
<td>BAD</td>
<td>{4,20}</td>
<td>40</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Doom QoS Level</th>
<th>Thread QoS Level</th>
<th>Memory QoS Level (kb)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GOOD</td>
<td>{7,10}</td>
<td>50</td>
</tr>
<tr>
<td>AVERAGE</td>
<td>{7,15}</td>
<td>30</td>
</tr>
<tr>
<td>BAD</td>
<td>{7,20}</td>
<td>20</td>
</tr>
</tbody>
</table>
Experiment - Qinna integration
Analysis

Dynamicity
- Allows dynamic QoS management
- Dynamic evaluation of QoS parameters
- Allows dynamic architecture reconfiguration

Reusability
- Component-based
- Separation policy/mechanism

Genericity
- Non QoS policy/mechanism/specification specific
- Each component has its own QoS mechanisms
- Management of heterogeneous QoS (real-time and non real-time)

Real-time aspects
- Admission testing can be omitted (hard real-time)
- Performance depends on the implementation (avoid multiple admission testing)
- Adaptation occurs in soft real-time
Conclusion

Summary
Qinna: component-based QoS architecture
Some preliminary experiments including various QoS mechanisms, QoS policies and QoS specifications

Benefits
Dynamicity
Reusability
Genericity

Future work
Integrate complex QoS management policies
Quantitative evaluation (LOC, memory, delay overhead, etc.)
Supporting tools