Qinna,
A Component-Based QoS Architecture for Embedded Systems

Jean-Charles Tournier
France Télécom R&D
Jeancharles.tournier@rd.francetelecom.com
Context - Mobile embedded systems

Several constraints
  Economic, ergonomic, mobility
More and more complex applications
  Multimedia=>QoS
Limited and dedicated resources
  CPU, memory, battery, network
Lead to dedicated systems
  Hardware (SoC, DSP) and software (OS, applications)

Need for open and flexible systems
  Add/Modify/Remove services or applications on the fly
Context - Embedded mobile systems

Dynamic and safe QoS management (resources level)

- Generic: independent of QoS policies
- Heterogeneous: management of heterogeneous QoS constraints
- Dynamic
- Self-configurable: dynamic evaluation of QoS requirements
- Safe: enforcement of QoS specification
- Reusable: reusability of QoS management policies
Context - CBSE

Component-Based Software Engineering (CBSE)
An easier way to build complex software by assembling, composing, software entities called components

CBSE provides
Adaptability, reusability, scalability

Large application domain
Distributed applications
  EJB, CCM, .NET, DCOM, OSGI
Embedded systems
  PECOS, VEST, Koala, Fractal/Think
Context - CBSE

QoS management

Genericity

- Predefined QoS Policies
  - EJB:persistence, Koala:memory, VEST: RT scheduling

Heterogeneity

Dynamicity

- Only for non-resource QoS level (e.g. security, transaction, persistence)
  - EJB, CCM

Self-configurability

Safety

- Only for non-dynamic QoS management
  - PECOS, VEST, Koala

Reusability

- Only for non-resource QoS level
  - EJB, CCM
Objective

To manage, at run time, resource QoS level in open component-based embedded systems

Generic, dynamic, heterogeneous, safe, self-configurable, reusable
Which component model?

Requirements

Simple
Extensible
Identification of components at run time
Fit embedded systems
Control over resources

The Fractal component model performs these requirements
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)
The Think Framework

Framework for component-based operating systems
Conforms to the Fractal model (Fractal’s implementation)
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.
How to manage QoS in components?

Composition of components leads to a contract

Four levels of contract:
  functional, pre/post conditions, synchronization, QoS
QoS contract management

- Concepts
  - Specification
    - Performance
      - GOOD
    - Importance
      - HIGH
    - Adaptation
      - GOOD
    - Mapping
      - GOOD
    - Admission
      - Testing
        - Sched_anal
          - (60%)
        - reserve
          - (60%)
    - Reservation
      - 25 f/sec
        - ?
    - Observation
      - improve_ressource
    - Maintenance
      - degrade_to
    - Adaptation

France Télécom R&D

Leuven - 30/08/2005 11
Qinna - Objectives summary

Objective
To manage QoS in components

Target
Embedded Systems

Component model
Fractal/Think
QoS may be managed through contracts
Specification, initialization, management

Software engineering concepts
Aspects separation (functional vs QoS)
Policy and mechanism separation
Qinna is an architecture defined through

A set of APIs

Component types

Abstract Data types

Inter-component relationships

Dynamic behavior
Example

- UI
- Video
- DOOM
- Threads
- Memory
- Sched

<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>40%</td>
<td>60</td>
</tr>
<tr>
<td>BAD</td>
<td>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>70%</td>
<td>50</td>
</tr>
<tr>
<td>AVERAGE</td>
<td>50%</td>
<td>30</td>
</tr>
<tr>
<td>BAD</td>
<td>25%</td>
<td>20</td>
</tr>
</tbody>
</table>
Example - Qinna integration
Analysis
Qinna vs Genericity

Answers
  Generic API
  Definition of component types (empty boxes)

Experiments
  Real-time (EDF, soft/hard RT, etc.) and multimedia systems
  Identification of each sub-contract and its associated QoS

Conclusions
  Useful for applications with several QoS levels (multimedia)
  Clear structuration of systems
    identification of each element and operation needed for a QoS management (RT)
Qinna vs Dynamicity - 1/3

Answers

- Identification of QoS contracts
- Integration of QoS contract management concepts

Experiments

- Evolution from a software point of view
  - Arrival/departure of a component
  - Variable required QoS
  - Modification of the importance order relation
- Evolution from a hardware point of view
  - Battery, fluctuating network, CPU
Qinna vs Dynamicity - 2/3

Costs
  Component granularity

Each sub-contract must be set up
Worst case (the last contract can not be set up)
  Set up and cancel of N-1 contracts
Qinna vs Dynamicity - 3/3

Costs

- Number of QoS levels of each component (Q)
- Number of QoS contracts managed by the QoSDomain (M)
- In the worst case the C contract needs 
  2N.Q.M operations
- Costs also depends on the implementation
  e.g. which contract must be set up first?

Conclusions

- Qinna fits targeted systems
  Limited number of components
  Hardware and software resources are well known
Qinna vs Self-configurability - 1/2

Answers

Contract attributes (reliable/unreliable)
Definition of the default QoS level
Identification of observer (observation policy)
Maintenance policy and mechanism

Experiments

Evaluation of required QoS level
  Fixed
  Fluctuating
Qinna vs Self-configurability - 2/2

Two ways

- Contract at the max level
  - Waste of resources
  - Max level may be unknown
- Contract at the min level
  - Contract violation
  - Delay due to violation
  - Maintenance operation
    - modify the contract of a step

Conclusions

- Depends on the outline of the required QoS
- Trade-off between number of maintenance operations, waste of resources and delay
- Creation of delay impacts the QoS policy
Conclusion

Context
Open component-based embedded systems

Qinna
A component-based QoS architecture

Component-based
Clear structuration of the system
Identification of each QoS operation and of its cost

Contract-based
Makes explicit the QoS relation between components
Easier to manage QoS by defining an envelope (c.f. real-time)
Trade-off between waste of resources and adaptation/observation/maintenance operations
Future works

Define a meta-model
   Automatic generation
   Allows to verify properties of the architecture

Extend the architecture to others non-functional properties
   Fault tolerance, security, etc.
   How to compose non-functional properties?

Extend the architecture to distributed and multiprocessors systems
   Single QoSDomain vs QoSDomain cooperation