Integration of Flexible Real-Time Scheduling Services in a LwCCM-Based Framework

Authors: O. Hachet, J.L. Gilbert, J. Chauvin (Thales)
M. Gonzales, J. Medina, P. Lopez (Univ Cantabria)
FRESCOR Project Presentation

- Overview and previous projects
- Project Goals
- FRSH programming model

LightWeight-CCM integration

- Main approach
- Timing requirements
- Components/contract association
- Deployment plan & components assembly
Overview and previous projects

FRESCOR

- Framework for Real-time Embedded Systems based on COntrRacts
- Project funded in part by European Union
- Consortium research project following:
  - FIRST: dedicated to flexible scheduling and contract-based techniques
  - COMPARE: CCM applied to RTE systems
  - OCERA: Real-time kernel and components

Objectives

- Develop enabling technology and infrastructure to use the most advanced techniques developed for real-time application
- Higher level programming model used together with RTE systems design methodology (from OS to application)
What we observe

Industrial products with real-time behaviour should be designed in the following way:

- WCET estimation should be realised
- The whole system doesn’t completely needs hard real-time constraints, hard real-time part is small
- Available resources has to be used in adequate manner
- Most of the time no real-time analysis is provided to test the system
  - Timing requirements are “proven” by testing
  - Hard real-time analysis is supposed to be too pessimistic
Real-time scheduling theory could be useful

- But, needs proper abstraction
- And has to be integrated in the design process

Proposed approach

- API has to be platform independent
- Uses advanced scheduling method coming from real-time theory
  - Built-in analysis
  - Minimum requirements could be guaranteed
- Higher level programming model used together with RTE systems design methodology
- Introduction of Component-based techniques
- Contract-based abstraction
- Resources protection
Specific objectives

- Contract model that specifies application requirements
  - required to be guaranteed
  - usable to increase quality of service
- Underlying implementation manages & enforces contracts
  - integrated resources (processor, network, power, multiprocessor, reconfigurable hardware)
- Adaptive QoS Manager
- Distributed transaction manager
- Performance analysis via simulation
- Component-based framework bridges the gap with design methods
  - tools allow independent analysis
  - tools calculate contract parameters
  - tools obtain timing properties of the overall system
- Test & evaluate on three application domains
FRESCOR Project Presentation
- Overview and previous projects
- Project Goals
- FRSH programming model

LightWeight-CCM integration
- Main approach
- Timing requirements
- Components/contract association
- Deployment plan & components assembly
Contract definition

Application

Contract
Application Requirements

Contract
Application Requirements

Negotiation

FRSH scheduler

Virtual resource
Copy of contract
Consumed resources
OS resources

Virtual resource
Copy of contract
Consumed resources
OS resources

Operating System
Contract-based scheduling

Contract specifies
- Minimum requirements for a given resource
- How to make use of any spare capacity

On-line and off-line acceptance tests

Spare resources are distributed according to importance and weight
- Statically or dynamically

Renegotiation is possible
Major features of FRESCOR contracts:

Coverage of application requirements
- mixture of hard and soft real-time

Platform independent API
- independent of OS

Independent of underlying scheduler
- Support for multiple resources
  - processors, networks
  - memory, energy

Ease of building advanced real-time applications
- by having time and timing requirements in the API
Contract negotiation
FRESCOR

- Core
- Dynamic reclamation
- Implementation Specific
- Shared resources
- Memory management
- Hierarchical scheduling
- Spare capacity sharing
- Energy management
- Distributed
- Feedback control
An API example

With OS API

Set priority
Create budget signal handler
create deadline signal handler
create budget timer
create deadline timer
while (true) {
    reset deadline timer
    set budget timer
    do useful things
    reset budget timer
    set deadline timer
    wait for next period
}

With FRSH API

Create contract with (C,T)
Negotiate the contract
while (true) {
    do useful things
    frsh_timed_wait
}
FRESCOR Project Presentation
- Overview and previous projects
- Project Goals
- FRSH programming model

LightWeight-CCM integration
- Main approach
- Timing requirements
- Components/contract association
- Deployment plan & components assembly
Framework main ideas

- Reusable components with passive operations
- Threads for executing the operations offered and managed by the container
- Connectors used for communication management
- FRESCOR management achieved by interception
Business code formulated as passive operations

Two kinds of operations can be executed by environment threads on a component:

- Activation operations: One Shot or Periodic
  - Formulated as ports offering “special” interfaces
- Invocations received in a facet. Different execution modes:
  - Synchronous or asynchronous
    - Defined at specification level (IDL)
    - Managed by the connector
  - Client controlled or Global Activity controlled
    - Defined in the deployment file
    - Managed by interceptors

```java
interface OneShotActivation{
    void run();
};
```

```java
interface PeriodicActivation{
    void update()
    raises(EndActivationException);
};
```
FRESCOR Project Presentation
- Overview and previous projects
- Project Goals
- FRSH programming model

LightWeight-CCM integration
- Main approach
- Timing requirements
- Components/contract association
- Deployment plan & components assembly
**Support the real-time model:**
- Assign scheduling attributes to invoking threads
- Differentiates invocations based on global activities

![Diagram showing the Interceptor system with Client and Server components and a reactive model with input and output stimuli IDs and scheduling attributes.]

Automatically generated according to deployment data

<table>
<thead>
<tr>
<th>InputStimID</th>
<th>OutputStimID</th>
<th>SchedAttrID</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>14</td>
<td>2</td>
</tr>
<tr>
<td>23</td>
<td>25</td>
<td>2</td>
</tr>
<tr>
<td>18</td>
<td>33</td>
<td>5</td>
</tr>
<tr>
<td>37</td>
<td>45</td>
<td>7</td>
</tr>
</tbody>
</table>

**SchedAttrService**
- SchedAttrList
- `addSchedAttr(..)`
- `bind(…)`
- `unbind(…)`
Concept of activity and stimulusId

Invocation of A.OperX with StimulusId = 10

StimulusId = 10

C.OperZ_1 is executed with StimulusId = 10 \(\Rightarrow\) Contract 1 (SchedAttr = 1)

C.OperZ_3 is executed with StimulusId = 20 \(\Rightarrow\) Contract 2 (SchedAttr = 2)
FRESCOR Project Presentation
- Overview and previous projects
- Project Goals
- FRSH programming model

LightWeight-CCM integration
- Main approach
- Timing requirements
- Components/contract association
- Deployment plan & components assembly
Keep the list of available Scheduling Attributes (vres in case of FRESCOR)

- Bind and unbind threads to the corresponding attributes (vres) to execute a method of the component.
Example: Track follower

- **OneShotActivation**
  - run()

- **PeriodicActivation**
  - update()

**TrackFollower** (Client)
- keyboardThread (200 ms)
- alarmMonitorThread
- trackerThread

**ServosController**
- controller
- ioCard
- alarmSource
- logger
- loggerThread (1000 ms)
- controlThread (5 ms)

**SoundGenerator**
- soundThread (125 ms)

**I_Control**
- setControlGains(…)
- goTo(newPosition)
- getPosition(): Position
- setThreshold(…)
- logTrajectory()
- cancelLogging()

**I_AnalogIO**
- getValue(inputLine): Float
- setValue(outputLine, value)

**I_Player**
- play(melody)
- manyPlay(melody)
- fail()
- alarm()

**I_Logger**
- log(type, message)
- getLastEvent(type, num)
- awaitEvent(type)
- Alarm()
**FRSH usage example**

**MPEG2 decoder showing spare capacity usage**

- **Bmax**: maximum budget to allocate to use spare capacity
- **Pmin**: Minimum period to enforce

- **Bmin**: minimum budget (execution time)
- **Pmax**: Maximum period to enforce (maybe equal to deadline)

**Frame Input**
- **Bmin**: 1ms
- **Bmax**: 3ms
- **Pmax**: 5ms
- **Pmin**: 7ms
- Time triggered

**Frame Decoder**
- **Bmin**: 1ms
- **Bmax**: 3ms
- **Pmax**: 7ms
- **Pmin**: 10ms
- Time triggered

**Frame Output**
- **Bmin**: 1ms
- **Bmax**: 3ms
- Event triggered

**FRSH + functional code**

Sensor ➔ Display
MPEG2 decoder showing spare capacity usage

Frame Input
- **Defined in CDP files**
- **Bmin:** 1ms, **Bmax:** 3ms, **Pmax:** 5ms, **Pmin:** 7ms

Frame Decoder
- **Bmin:** 1ms, **Bmax:** 3ms, **Pmax:** 7ms, **Pmin:** 10ms

Frame Output
- **Bmin:** 1ms, **Bmax:** 3ms

Functional code
- Time triggered
- Periodic Activation port

FRSH code in non-functional services
- Input StimId
- output StimId
- Periodic Activation port

Event triggered
- OneShot Activation port
FRESCOR Project Presentation
- Overview and previous projects
- Project Goals
- FRSH programming model

LightWeight-CCM integration
- Main approach
- Timing requirements
- Components/contract association
- Deployment plan & components assembly
Deployment Plan

Includes the aspects that are application-dependent: i.e., the scheduling parameters

```xml
<DnCedm:DeploymentPlan>
  ...
  <instance name="theSpeaker" node="node1" />
  <!-- Property to configure the Periodic Activation -->
  <property name="soundThread">
    <value>
      <periodicActivationProperty period="0.005" schedAttrId="1"
    </value>
  </property>
  <!-- Property to configure invocation modes of operations -->
  <environmentProperty portname="I_Player_Port"
    operation="play">
    <executionData inputStimId="10" outputStimId="10"
      schedAttrId="2"
      executionMode="TransactionControlled"/>
    <executionData inputStimId="50" outputStimId="60"
      schedAttrId="3"
      executionMode="ClientControlled"/>
  </environmentProperty>
  ...
</DnCedm:DeploymentPlan>

<DnCedm:TargetDataModel>
  ...
  <node name="node1" />
  <schedulingAttribute id="1">
    <value>
      <contract contractId="1"
        contractParams="..."/>
    </value>
  </schedulingAttribute>
  <schedulingAttribute id="2">
    <value>
      <contract contractId="2"
        contractParams="..."/>
    </value>
  </schedulingAttribute>
  ...
</DnCedm:TargetDataModel>
```
Conclusion and Future Work

**Conclusion**
- FRSH API allows to encapsulate several scheduling policy
- New programming model leverage development of Real-time application with soft and hard constraints
- Used together with Components technology permits to modelize the behaviour of an RT application
- LightweightCCM enable RT constraints enforcement via contract definition and activities

**Future work**
- Distribution of contract techniques on network
- Contract parameters evaluation via simulation tools
- Reconfiguration via on-line scheduling analysis
- Use-case assessment