Implementation of a Flexible Real-Time Scheduling Middleware Based on Contracts

Implementación de un middleware de planificación flexible de tiempo real basado en contratos

Miguel Telleria de Esteban
directed by Michael González Harbour

Universidad de Cantabria – Computadores y Tiempo Real

30 Oct. 2008

Work done under FRESCOR project (FP6/2005/IST/5-034026) funded in part by the European Union’s Sixth Framework Programme.
This document, *Implementation of a Flexible Real-Time Scheduling Middleware Based on Contracts* by Miguel Telleria from *Computadores y Tiempo Real, Univ. de Cantabria* \(^a\) is licensed under a Creative Commons Attribution ShareAlike 3.0 license \(^b\).

\(^a\)http://www.ctr.unican.es  
\(^b\)http://creativecommons.org/licenses/by-sa/3.0/legalcode/

This means that you are free:

- **To share**, to copy, distribute and transmit the work.
- **To remix**, to adapt the work (even for commercial purposes).

Under the following **conditions**:

- **Attribution** You must give the original author credit.
- **ShareAlike** If you alter, transform, or build upon this work, you may distribute the resulting work only under the same, similar or a compatible license.

Any of the above conditions can be waived if you get permission from the copyright holder.

Original editable \LaTeX{} content should be available from the same source of the binary document or from the copyright holder possibly upon request.
Outline

1. Background and motivation
   - Flexible scheduling
   - FIRST and FRESCOR projects

2. Objectives

3. Work done
   - Global architecture
   - Migration process
   - New additions

4. Conclusions
   - Status of FRSH
   - Conclusions and lessons learned
Outline

1. **Background and motivation**
   - Flexible scheduling
   - FIRST and FRESCOR projects

2. **Objectives**

3. **Work done**
   - Global architecture
   - Migration process
   - New additions

4. **Conclusions**
   - Status of FRSH
   - Conclusions and lessons learned
Embedded Real-Time systems

Definitions

- **Embedded systems**: Computers considered an integral part of a larger system that they control and/or monitor.
- **Real Time system**: Having results on time is as important as the results themselves.

Goal: Time predictability

- At hardware level (detection of events, transmission of actions).
- At operating system and network level (context switch, timed services, interrupt latency, network Tx/Rx...).
- At the application level (through analysis techniques).
Embedded Real-Time systems

Definitions

- **Embedded systems**: Computers considered an integral part of a larger system that they control and/or monitor.
- **Real Time system**: Having results on time is as important as the results themselves.

Goal: Time predictability

- At hardware level (detection of events, transmission of actions).
- At operating system and network level (context switch, timed services, interrupt latency, network Tx/Rx...).
- At the application level (through analysis techniques).
**Hard real time vs Flexible real time**

**Traditional real-time**
- Worst case response.
  - Static resource allocation.
    - Single mode
  - No time protection.
  - No adaptation to load change.

**Flexible real-time**
- Worst case response + QoS
  - Dynamic resource allocation
    - Multiple mode
  - Time protection.
  - Load change adaptation:
    - **Spare capacity**: (mode change)
    - **Dynamic reclamation**: (execution)

**Benefits of Flexible Scheduling**
- Maximise resource usage.
- Integration of heterogeneous resource requirements.
- Real time theory implicitly integrated in system.
Flexible scheduling

Hard real time vs Flexible real time

Traditional real-time
- Worst case response.
- Static resource allocation.
  - Single mode
- No time protection.
- No adaptation to load change.

Flexible real-time
- Worst case response + QoS
- Dynamic resource allocation
  - Multiple mode
- Time protection.
- Load change adaptation:
  - Spare capacity: (mode change)
  - Dynamic reclamation: (execution)

Benefits of Flexible Scheduling
- Maximise resource usage.
- Integration of heterogeneous resource requirements.
- Real time theory implicitly integrated in system.
Hard real time vs Flexible real time

Traditional real-time
- Worst case response.
- Static resource allocation.
  - Single mode
- No time protection.
- No adaptation to load change.

Flexible real-time
- Worst case response + QoS
- Dynamic resource allocation
  - Multiple mode
- Time protection.
- Load change adaptation:
  - Spare capacity: (mode change)
  - Dynamic reclamation: (execution)

Benefits of Flexible Scheduling
- Maximise resource usage.
- Integration of heterogeneous resource requirements.
- Real time theory implicitly integrated in system.
Hard real time vs Flexible real time

**Traditional real-time**
- Worst case response.
- Static resource allocation.
  - Single mode
- No time protection.
- No adaptation to load change.

**Flexible real-time**
- Worst case response + QoS
- Dynamic resource allocation
  - Multiple mode
- Time protection.
- Load change adaptation:
  - **Spare capacity**: (mode change)
  - **Dynamic reclamation**: (execution)

**Benefits of Flexible Scheduling**
- Maximise resource usage.
- Integration of heterogeneous resource requirements.
- Real time theory implicitly integrated in system.
Hard real time vs Flexible real time

**Traditional real-time**
- Worst case response.
- Static resource allocation.
  - Single mode
- No time protection.
- No adaptation to load change.

**Flexible real-time**
- Worst case response + QoS
- Dynamic resource allocation
  - Multiple mode
- Time protection.
- Load change adaptation:
  - **Spare capacity**: (mode change)
  - **Dynamic reclamation**: (execution)

**Benefits of Flexible Scheduling**
- Maximise resource usage.
- Integration of heterogeneous resource requirements.
- Real time theory implicitly integrated in system.
Flexible Scheduling Execution

- **Mode 1:** $C_A = 3 \quad T_A = 6 \quad C_B = 1 \quad T_B = 3$
- **Mode 2:** $C_A = 1 \quad T_A = 3 \quad C_B = 3 \quad T_B = 6$
Contract model

Requirements specified in contract

- Minimum capacity: \( \text{budget}_{\text{min}} (C) \), \( \text{period}_{\text{max}} (T) \) and deadline (D).
- Extra capacity: Range or discrete values of (C, T, D).
- Task model: Job-based, continuous, background.
- Importance and weight as criteria for spare capacity distribution.
- Critical sections (with their WCET) on shared objects.

Negotiation and binding
Contract model

Requirements specified in contract

- Minimum capacity: $\text{budget}_{\text{min}}$ (C), $\text{period}_{\text{max}}$ (T) and deadline (D).
- Extra capacity: Range or discrete values of (C, T, D).
- Task model: Job-based, continuous, background.
- Importance and weight as criteria for spare capacity distribution.
- Critical sections (with their WCET) on shared objects.

Negotiation and binding
FIRST project

- 4 academic partners: UC, Pisa, York, Mälardalen (Suecia)
- Outcome: **FSF: First Scheduling Framework**

**FSF Achievements**

- Proved the contract model on MaRTE-OS (fixed priorities) and S.Ha.R.K (EDF).
- Resources: CPU and network (MaRTE-OS only).
- Spare capacity distribution for CPU only.
- Dynamic reclamation (S.Ha.R.K. only).
- Utilisation-based analysis.
- Hierarchical scheduling (S.Ha.R.K. only).

**FSF weak aspects**

- Implemented only in academic operating systems.
- Prototype approach.
FIRST and FRESCOR projects

FIRST project

- 4 academic partners: UC, Pisa, York, Mälardalen (Suecia)
- Outcome: **FSF: First Scheduling Framework**

**FSF Achievements**

- Proved the contract model on MaRTE-OS (fixed priorities) and S.Ha.R.K (EDF).
- Resources: CPU and network (MaRTE-OS only).
- Spare capacity distribution for CPU only.
- Dynamic reclamation (S.Ha.R.K. only).
- Utilisation-based analysis.
- Hierarchical scheduling (S.Ha.R.K. only).

**FSF weak aspects**

- Implemented only in academic operating systems.
- Prototype approach.
FIRST and FRESCOR projects

FRESCOR project

QoS Adapter
MyCCM Component Framework
distributed Transaction layer
encapsulates

Response-time analysis
FRSH Middleware
designed on top of
integrated in

FSF evolves into FRSH

Execution Platforms
OSE AQuoSA PaRTiKle MaRTE OS RapiTime
simulator
FRSH
Middleware
RapiTime
WCET tool

QoS Adapter
MyCCM Component Framework
distributed Transaction layer
encapsulates
Response-time analysis
FRSH Middleware
designed on top of
integrated in
FSF evolves into FRSH

OSE AQuoSA PaRTiKle MaRTE OS RapiTime
simulator
FRSH
Middleware
RapiTime
WCET tool

QoS Adapter
MyCCM Component Framework
distributed Transaction layer
encapsulates
Response-time analysis
FRSH Middleware
designed on top of
integrated in
FSF evolves into FRSH

OSE AQuoSA PaRTiKle MaRTE OS RapiTime
simulator
FRSH
Middleware
RapiTime
WCET tool

QoS Adapter
MyCCM Component Framework
distributed Transaction layer
encapsulates
Response-time analysis
FRSH Middleware
designed on top of
integrated in
FSF evolves into FRSH

OSE AQuoSA PaRTiKle MaRTE OS RapiTime
simulator
FRSH
Middleware
RapiTime
WCET tool

QoS Adapter
MyCCM Component Framework
distributed Transaction layer
encapsulates
Response-time analysis
FRSH Middleware
designed on top of
integrated in
FSF evolves into FRSH

OSE AQuoSA PaRTiKle MaRTE OS RapiTime
simulator
FRSH
Middleware
RapiTime
WCET tool
FIRST and FRESCOR projects

Making life easier for Real Time developers

**FRSH relieves the application from:**

- Computation of capacity distribution.
- Operating system scheduling policies.
- Enforcement of assigned capacity, by providing time isolation to application components.

**FRSH Facilitates:**

- Implementation of QoS policy to maximize resource usage.
- Guaranteed transactions in distributed systems.
- Encapsulate time requirements in Software Components or other abstractions.
FIRST and FRESCOR projects

Making life easier for Real Time developers

FRSH relieves the application from:

- Computation of capacity distribution.
- Operating system scheduling policies.
- Enforcement of assigned capacity, by providing time isolation to application components.

FRSH Facilitates:

- Implementation of QoS policy to maximize resource usage.
- Guaranteed transactions in distributed systems.
- Encapsulate time requirements in Software Components or other abstractions.
Outline

1. Background and motivation
   - Flexible scheduling
   - FIRST and FRESCOR projects

2. Objectives

3. Work done
   - Global architecture
   - Migration process
   - New additions

4. Conclusions
   - Status of FRSH
   - Conclusions and lessons learned
Objectives of this work

- Evolution of FRSH-ADS for CPU’s during the FRESCOR project on MaRTE-OS.

### Migration tasks

1. Extend FSF to multiple execution platforms
   1. Define a neutral platform API.
   2. Port FSF to this platform.
2. Port FSF to the new FRSH API.

### New additions

1. Define a pluggable interface for scheduling analysis.
2. Contract group negotiation.
3. Time protection for shared objects.
Objectives of this work

- Evolution of FRSH-ADS for CPU’s during the FRESCOR project on MaRTE-OS.

Migration tasks

1. Extend FSF to multiple execution platforms
   1. Define a neutral platform API.
   2. Port FSF to this platform.
2. Port FSF to the new FRSH API.

New additions

1. Define a pluggable interface for scheduling analysis.
2. Contract group negotiation.
3. Time protection for shared objects.
Outline

1. Background and motivation
   - Flexible scheduling
   - FIRST and FRESCOR projects

2. Objectives

3. Work done
   - Global architecture
   - Migration process
   - New additions

4. Conclusions
   - Status of FRSH
   - Conclusions and lessons learned
API agreed between FRSH platform providers.

- POSIX Neutral names for OS services.
- Scope: FRSH.
API agreed between FRSH platform providers.

- POSIX Neutral names for OS services.

- Scope: FRSH.
API agreed between FRSH platform providers.

POSIX Neutral names for OS services.

Scope: FRSH.
Application Defined Scheduling

- Event driven
- Interferes with native scheduler
- Threads ADS or non ADS

Internal Status:
- task queue
- event queue
- ...

Native Scheduler

Operating System

Invoke

Init
New Thread
Terminate
Ready
Block
Explicit Call
Signal
Timeout
...

Return Scheduling Actions

Interactive Call
Global architecture

Application Defined Scheduling

- Event driven
- Interferes with native scheduler
- Threads ADS or non ADS

Operating System

Native Scheduler

Application Defined Scheduler

Init
New Thread
Terminate
Ready
Block
Explicit Call
Signal
Timeout
...

Internal Status:
- task queue
- event queue
- ...

Return Scheduling Actions
Invoke

Invoke
Application Defined Scheduling

- Event driven
- Interferes with native scheduler
- Threads ADS or non ADS
Global architecture

FRSH architecture

- Application level
- FRSH
- API Front-End
- thread-specific API parameters
- explicit call
- block
- new_thread
- unblock
- signal
- App
- Scheduler
- service, thread
- FSA plugin
- scenarios
- ANALYSIS data
- thread data
- vres data
- General data
- RUN TIME data

FOSA

Operating System
Porting to new FRSH API

New FRSH API and Platform

- Service thread now scheduled by FRSH
  - **Rationale**: PaRTiKle and OSE lack native sporadic server support.
- **server** term renamed to **vres**
  - **Rationale**: More explicit and less ambiguous meaning.
- Abstraction of time types in FRSH and FOSA
  - **Rationale**: Time representation platform specific (64/32/16bit, struct...).
- Separate API functions for non-FRSH threads
  - **Rationale**: Application designer must give them special treatment.

Code rewriting

- Follow FSF decisions (time enforcement, callbacks)
  - More modularity (intermediate functions, abstract types)
  - Document all global variables in the code
- Use a version control system (subversion)
  - Remove obsoleted commented code.
  - Allows parallel development while preserving a functional version.
Porting to new FRSH API

New FRSH API and Platform

- Service thread now scheduled by FRSH
  - **Rationale**: PaRTiKle and OSE lack native sporadic server support.
- **server** term renamed to **vres**
  - **Rationale**: More explicit and less ambiguous meaning.
- Abstraction of time types in FRSH and FOSA
  - **Rationale**: Time representation platform specific (64/32/16bit, struct...).
- Separate API functions for non-FRSH threads
  - **Rationale**: Application designer must give them special treatment.

Code rewriting

- Follow FSF decisions (time enforcement, callbacks)
  - More modularity (intermediate functions, abstract types)
  - Document all global variables in the code
- Use a version control system (subversion)
  - Remove obsoleted commented code.
  - Allows parallel development while preserving a functional version.
FSA plugins

Prospective scenario

- vres A
- Contract 1
- shared object
- priorities
- overhead model

Running scenario

- vres A
- Contract 1
- shared object
- priorities
- overhead model

- apply changes
- synchronize
- analysis computations

contract changes
additions and removals
renegotiate contract
add new contract

- assign priorities
- assign ceilings
- schedulability test
- spare capacity distribution

Scheduling Analysis API

New additions

Contract

vres

A

Contract

1

Contract

2

vres

C

Contract

3

Prospective scenario

vres A

CA1

vres A

CA1
Protected shared objects

- Vres A
- Vres B
- Shared object
- WCET A
- WCET B

- Blocked
- Allowed to pass
- Rollback
- WCET exceeded
1. Background and motivation
   - Flexible scheduling
   - FIRST and FRESCOR projects

2. Objectives

3. Work done
   - Global architecture
   - Migration process
   - New additions

4. Conclusions
   - Status of FRSH
   - Conclusions and lessons learned
Status of FRSH

Contributions

Achievements

- FRSH has consolidated FSF functionality.
  - Now supported on several platforms (and others may join).
  - Using new FRSH API
- A pluggable scheduling analysis interface has been defined.
  - Original FSF utilisation-based analysis has been extended to multiple changes.
  - FRSH now supports response-time analysis.
- Time protection for critical section is provided.
Status of FRSH

Contributions

**Achievements**

- FRSH has consolidated FSF functionality.
  - Now supported on several platforms (and others may join).
  - Using new FRSH API
- A pluggable scheduling analysis interface has been defined.
  - Original FSF utilisation-based analysis has been extended to multiple changes.
  - FRSH now supports response-time analysis.
- Time protection for critical section is provided.
Contributions

Achievements

- FRSH has consolidated FSF functionality.
  - Now supported on several platforms (and others may join).
  - Using new FRSH API
- A pluggable scheduling analysis interface has been defined.
  - Original FSF utilisation-based analysis has been extended to multiple changes.
  - FRSH now supports response-time analysis.
- Time protection for critical section is provided.
**Left to do**

### Small issues
- Explicit functions for external threads, e.g. Ada tasks
- Support for synchronised workload.
- POSIX sporadic server corrections.
- Front-End support for group negotiations.

### Medium complexity
- Improvement in shared objects time protection.

### Large complexity
- Hierarchical scheduling
- Stability time
- Industrial validation
Left to do

Small issues
- Explicit functions for external threads, e.g. Ada tasks
- Support for synchronised workload.
- POSIX sporadic server corrections.
- Front-End support for group negotiations.

Medium complexity
- Improvement in shared objects time protection.

Large complexity
- Hierarchical scheduling
- Stability time
- Industrial validation
Status of FRSH

Left to do

Small issues

- Explicit functions for external threads, e.g. Ada tasks
- Support for synchronised workload.
- POSIX sporadic server corrections.
- Front-End support for group negotiations.

Medium complexity

- Improvement in shared objects time protection.

Large complexity

- Hierarchical scheduling
- Stability time
- Industrial validation
Conclusions and lessons learned

Conclusions

- FRSH is now more mature, documented and maintainable but it is still work in progress.
- It is starting to be used as a base for other works.

Lessons learned

- Substantial effort has been spent in non business tasks (renaming, refactoring with abstract data types).
- We need to generate complete documentation for newcomers.
- Extra effort in documenting and making source code maintainable pays off later.
Conclusions and lessons learned

Conclusions

- FRSH is now more **mature, documented and maintainable** but it is still **work in progress**.
- It is starting to be used as a base for other works.

Lessons learned

- Substantial effort has been spent in non business tasks (renaming, refactoring with abstract data types).
- We need to generate complete documentation for newcomers.
- Extra effort in documenting and making source code maintainable pays off later.
Conclusions and lessons learned

Conclusions

- FRSH is now more **mature, documented and maintainable** but it is still **work in progress**.
- It is starting to be used as a base for other works.

Lessons learned

- Substantial effort has been spent in non business tasks (renaming, refactoring with abstract data types).
- We need to generate complete documentation for newcomers.
- Extra effort in documenting and making source code maintainable pays off later.
Conclusions and lessons learned

Conclusions

- FRSH is now more mature, documented and maintainable but it is still work in progress.
- It is starting to be used as a base for other works.

Lessons learned

- Substantial effort has been spent in non business tasks (renaming, refactoring with abstract data types).
- We need to generate complete documentation for newcomers.
- Extra effort in documenting and making source code maintainable pays off later.
Conclusions and lessons learned

Conclusions
- FRSH is now more mature, documented and maintainable but it is still work in progress.
- It is starting to be used as a base for other works.

Lessons learned
- Substantial effort has been spent in non business tasks (renaming, refactoring with abstract data types).
- We need to generate complete documentation for newcomers.
- Extra effort in documenting and making source code maintainable pays off later.
Thank you for your attention
Questions?