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.

Copyright notice

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

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.

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 implicitely 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 implicitely 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 implicitely 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

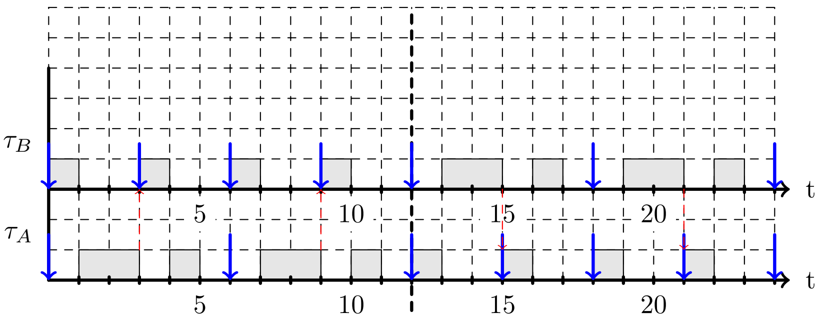
- Worst case response + QoS
- Dynamic resource allocation
  - Multiple mode
- Time protection.
- Load change adaptation:
  - **Spare capacity**: (mode change)
  - **Dynamic reclamation**: (execution)

Benefits of Flexible Scheduling

- Maximise resource usage.
- Integration of heterogeneous resource requirements.
- Real time theory implicitely integrated in system.

Flexible Scheduling Execution

- **Mode 1**: $C_A = 3$ $T_A = 6$ $C_B = 1$ $T_B = 3$
- **Mode 2**: $C_A = 1$ $T_A = 3$ $C_B = 3$ $T_B = 6$



Contract model

Requirements specified in contract

- Minimum capacity: $budget_{min}$ (C), $period_{max}$ (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.

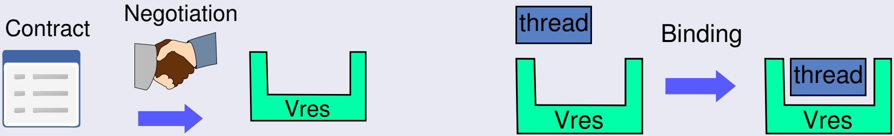
Negotiation and binding

Contract model

Requirements specified in contract

- Minimum capacity: $budget_{min}$ (C), $period_{max}$ (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 project

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

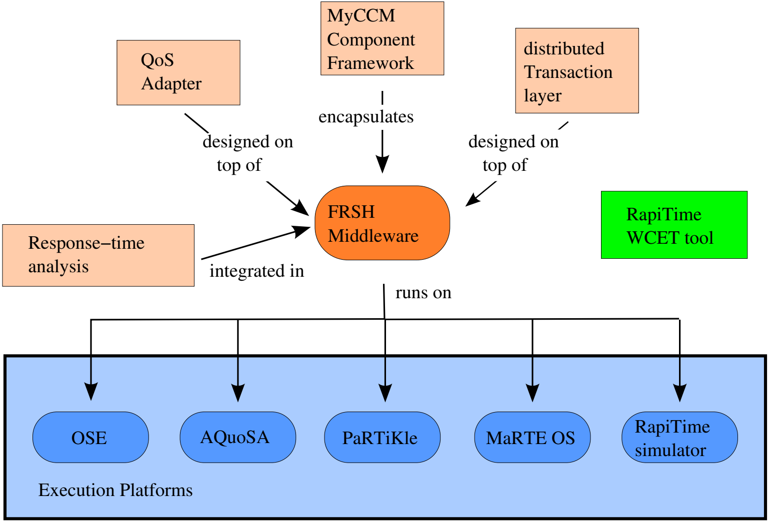
FSF Achievements

- 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.

FRESCOR project



FSF evolves into FRSH

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.

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

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

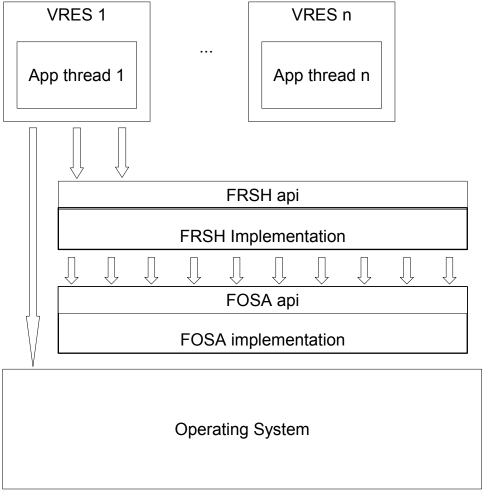
1. Extend FSF to multiple execution platforms
   1. Define a neutral platform API.
   2. Port FSF to this platform.
2. Port FSF to the new FRSH API.

New additions

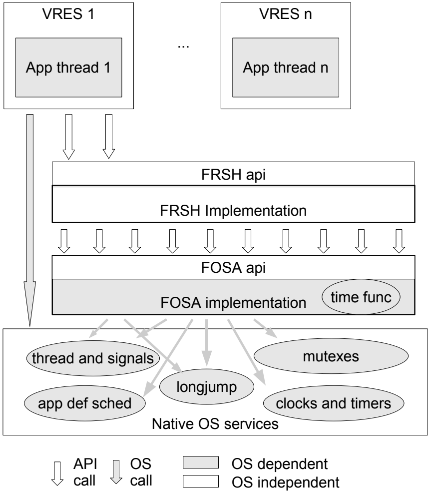
1. Define a pluggable interface for scheduling analysis.
2. Contract group negotiation.
3. Time protection for shared objects.

Outline

Global Picture

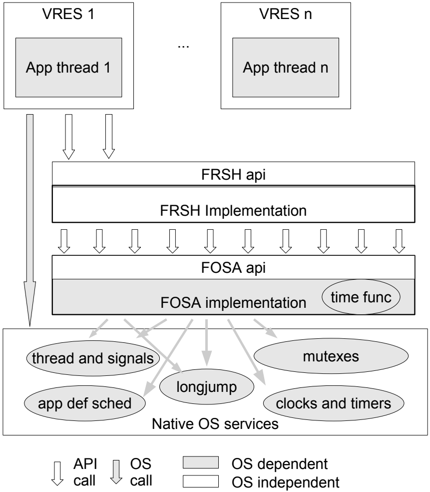


FOSA layer



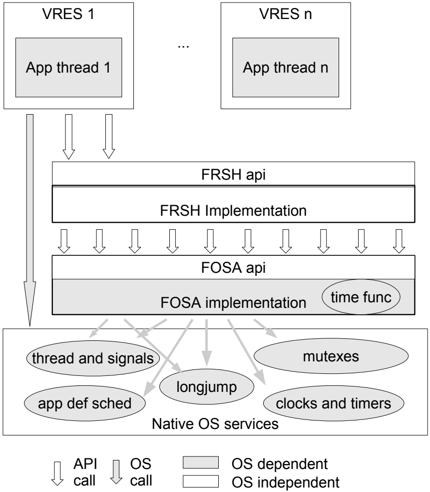
- API agreed between FRSH platform providers.
- POSIX Neutral names for OS services.
- Scope: FRSH.

FOSA layer



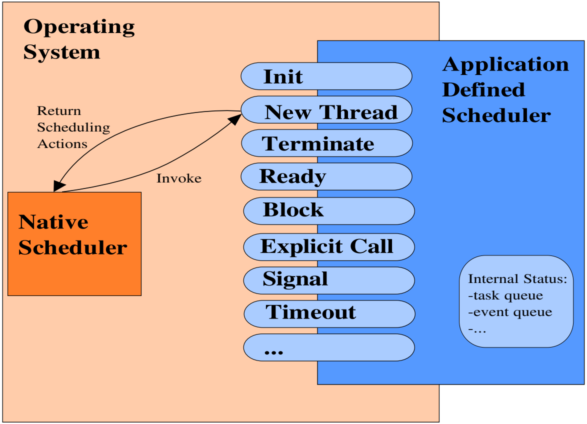
- API agreed between FRSH platform providers.
- POSIX Neutral names for OS services.
- Scope: FRSH.

FOSA layer



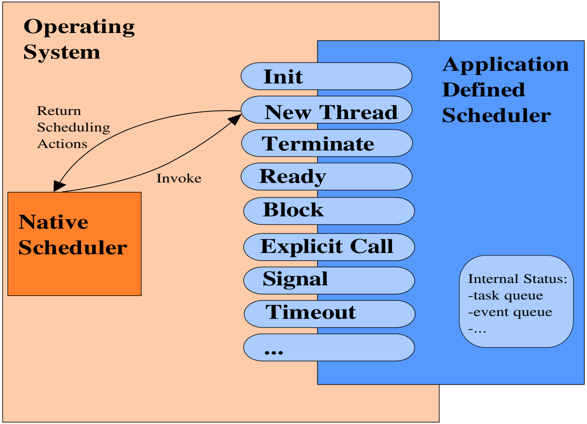
- API agreed between FRSH platform providers.
- POSIX Neutral names for OS services.
- Scope: FRSH.

Application Defined Scheduling



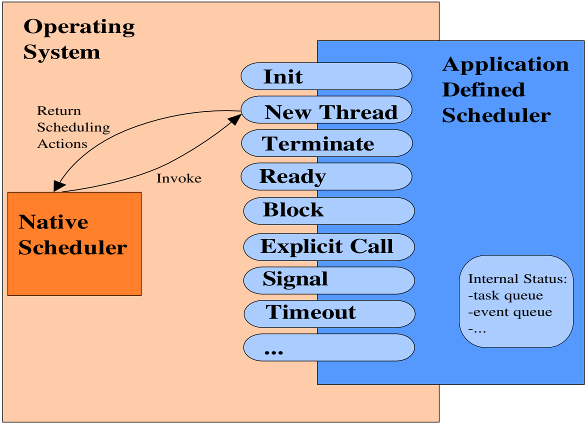
- Event driven
- Interferes with native scheduler
- Threads ADS or non ADS

Application Defined Scheduling



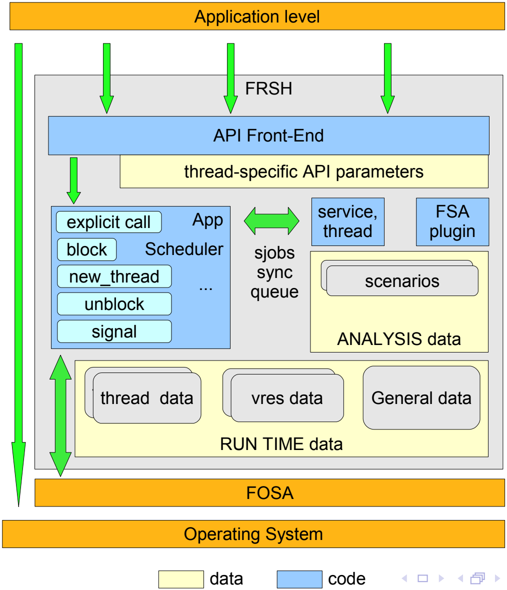
- Event driven
- Interferes with native scheduler
- Threads ADS or non ADS

Application Defined Scheduling



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

FRSH architecture



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

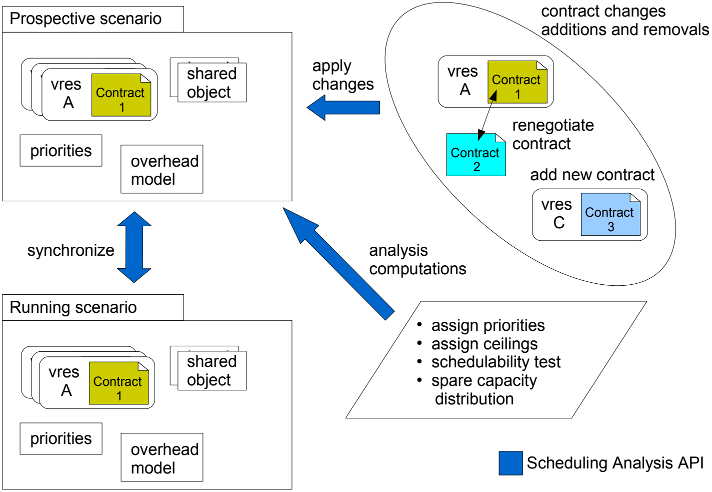
New FRSH API and Platform

- 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.

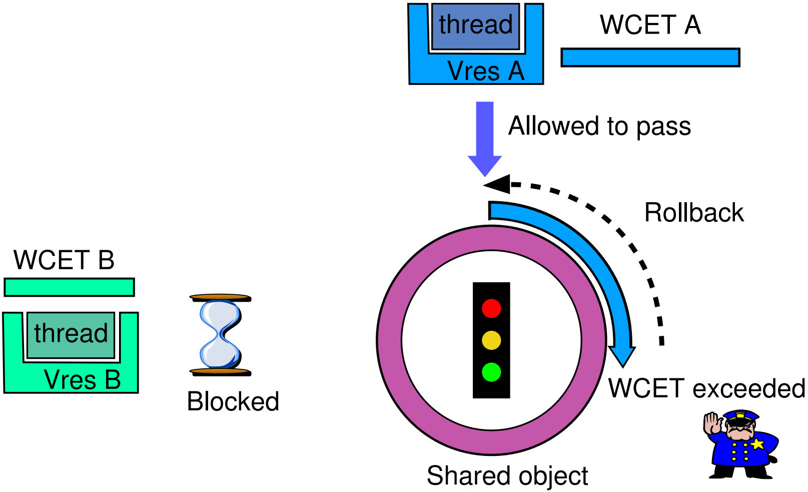
Code rewriting

- 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
apply changes
contract changes
additions and removals
vres A
Contract 1
Contract 2
renegotiate contract
add new contract
vres C
Contract 3
synchronize
analysis computations
Running scenario
vres A
Contract 1
shared object
priorities
overhead model
assign priorities
assign ceilings
schedulability test
spare capacity distribution
Scheduling Analysis API

Protected 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

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.

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

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.

Thank your for your attention
Questions?