Automatic Selection of Feasibility Tests With the Use of AADL Design Patterns

V. Gaudel†, F. Singhoff†, A. Plantec†, S. Rubini†
P. Dissaux*, J. Legrand*

†University of Brest/UBO, LISyC, France
*Ellidiss Technologies, France

24 may 2011
Case Study

- Simplified car system in AADL
- 3 functions: Headlights, windshield wiper and ESP control
- 12 threads (3 control threads)
- Data port communication
- Thread’s period: 30 ms
- Thread’s capacity: 2 ms
- Mono-processor
Motivations

- How to ensure safety of critical real-time systems?
- Multiple approaches: simulation, model checking, **analytical methods**, etc.

Real-time scheduling theory applicability difficulties

- Many methods specific to a restricted set of systems
- Need to select adequate methods
- Requires high level of expertise
- Unused in many practical cases
How to enforce real-time scheduling theory applicability?

- Automatisation of feasibility tests selection
- Modeling of relationships between architectural models in AADL and real-time scheduling analysis.
- Definition of real-time design patterns corresponding to a known set of feasibility tests.
- What are real-time design patterns, how to model and use them?
Outline

1. Feasibility tests and real-time design patterns
2. Method from user’s point of view
3. Design Patterns Modeling
4. Feasibility Tests Selection Algorithm
5. Evaluation
6. Discussion
Outline

1. Feasibility tests and real-time design patterns
2. Method from user’s point of view
3. Design Patterns Modeling
4. Feasibility Tests Selection Algorithm
5. Evaluation
6. Discussion
Schedulability analysis of critical systems: feasibility tests

Real-time system model:
- For each task $i$
- Deadline: $D_i$
- Capacity: $C_i$
- Period: $P_i$

Hypothesis
1. Periodic, synchronous and independent threads
2. Preemptive EDF or LLF Scheduling protocol

$$U = \sum_{i=1}^{n} \frac{C_i}{P_i} \leq 1$$

Necessary and sufficient condition if $\forall i : D_i = P_i$. If $\exists i : D_i < P_i$, then $\sum_{i=1}^{n} \frac{C_i}{D_i} \leq 1$ is a sufficient condition, and $\sum_{i=1}^{n} \frac{C_i}{P_i} \leq 1$ is a necessary condition.
**Definition Real-time Design patterns**

1. Based on inter-threads communication and synchronization paradigms.
2. Defined by a set of constraints on architectures.
3. Corresponding to a known number of cases for feasibility tests selection.

**Analysable performance criteria:**

- Worst case thread response times.
- Bounds on the thread waiting time due to data access.
- Deadlocks and priority inversions due to data access.
- Memory footprint analysis.
Design patterns description

1. Synchronous Data flow:
   Data port communication paradigm

2. Ravenscar:
   Shared data communication paradigm

3. Blackboard:
   ARINC 653, reader/writer communication protocol

4. Queued Buffer:
   Producer-consumer communication paradigm

5. Unplugged:
   No communication or synchronization between threads
Design patterns description

1. Synchronous Data flow:
   Data port communication paradigm

2. Ravenscar:
   shared data communication paradigm

3. Blackboard:
   ARINC 653, reader/writer
   communication protocol

4. Queued Buffer:
   producer-consumer communication
   paradigm

5. Unplugged:
   No communication or synchronization
   between threads

Synchronous Data-Flow

- R1 All threads are periodic
- R5’ No buffer
- R5” No data component
- R6 Data sharing protocol is sampled,
  immediate or delayed timing
- R7 No hierarchical scheduler: no shared
  address spaces between processors
Design patterns description

1. Synchronous Data flow:
   Data port communication paradigm

2. Ravenscar:
   Shared data communication paradigm

3. Blackboard:
   ARINC 653, reader/writer communication protocol

4. Queued Buffer:
   Producer-consumer communication paradigm

5. Unplugged:
   No communication or synchronization between threads

Ravenscar

R8 All tasks are periodic or sporadic
R9' At least one data component
R9'' No buffer
R10 For each data, there are, at least, two connected threads
R11 Allowed protocols: PCP, PIP, IPCP
R12 If PCP or IPCP are used, data's Ceiling priority must be superior to all dependent task's priority
R13 If PIP is used, dependent tasks cannot be connected to other resources
Outline

1. Feasibility tests and real-time design patterns
2. Method from user’s point of view
3. Design Patterns Modeling
4. Feasibility Tests Selection Algorithm
5. Evaluation
6. Discussion
Method from user’s point of view

Feasibility tests selection tool

Design

Architecture analysis

Automatic selection of feasibility tests

Feasibility tests application

Cheddar
Feasibility tests selection approach needs

- Is the model compliant to a design pattern?
- If not, how important are the modifications to become compliant?
- If it is, what is the list of relevant feasibility tests?
- Is there other potential lists and how important are the modifications to select them?
- Are the selected feasibility tests able to prove the schedulability?
- Is the system schedulable?
Outline

1. Feasibility tests and real-time design patterns
2. Method from user’s point of view
3. Design Patterns Modeling
4. Feasibility Tests Selection Algorithm
5. Evaluation
6. Discussion
Design Patterns Modeling

Use of EXPRESS to model our patterns
→ Use to model types and entities (Cheddar meta-model)
→ Enables to defined OCL like constraints

We enrich this meta-model for our design patterns

1. Hardware Context (environment mono-processor or multi-processors for instance)
2. Design patterns constraints
3. Sets of cases for feasibility tests selection (one per design pattern)
Part of Cheddar Meta-Model

Modeling of Tasks within Cheddar meta-model in EXPRESS

```
SCHEMA Tasks;

... TYPE Tasks_Type = ENUMERATION
      OF ( Periodic_Type, Aperiodic_Type, Sporadic_Type, Poisson_Type, Parametric_Type );
END_TYPE;

ENTITY Generic_Task
  ABSTRACT SUPERTYPE
  SUBTYPE OF ( Generic_Object );
  ...
  Cpu_Name : STRING;
  Address_Space_Name : STRING;
  Capacity : Natural;
  Deadline : Natural;
  ...
END_ENTITY;

ENTITY Periodic_Task
  SUBTYPE OF ( Generic_Task );
  Period : Natural;
  Jitter : Natural;
  ...
END_ENTITY;
```
Synchronous Data Flow Modeling in EXPRESS

R1  All tasks are periodic

All tasks are periodic

```plaintext
RULE all_tasks_are_periodic FOR ( generic_task );
WHERE
  R1 : SIZEOF ( QUERY ( t <=* generic_task | NOT ( 'TASKS.PERIODIC_TASK' IN TYPEOF ( t ) ) ) ) = 0;
END_RULE;
```

- Rule applied to all generic_task instances
- Use of set operators and SQL like queries
- Is true when the size of the set of non-periodic tasks within the totality of system’s tasks is equal to 0
- Each applicability constraint is modeled that way
Mono-processor environment Modeling in EXPRESS

R2 : Authorized scheduling protocols : fixed priorities, EDF, RM, DM
R3 : Preemptive or not preemptive
R4 : Quantum must be equal to 0

Data sharing protocol

```verbatim
ENTITY Mono_Processor_Environment
  SUBTYPE OF ( Environment );
WHERE
  R2 : ( 'SCHEDULERS. EARLIEST_DEADLINE_FIRST_PROTOCOL' IN TYPEOF ( SELF\Environment. scheduler ) ) OR
       ( 'SCHEDULERS. RATE_MONOTONIC_PROTOCOL' IN TYPEOF ( SELF\Environment. scheduler ) ) OR
       ( 'SCHEDULERS. DEADLINE_MONOTONIC_PROTOCOL' IN TYPEOF ( SELF\Environment. scheduler ) ) OR
       ( 'SCHEDULERS. POSIX_1003_HIGHEST_PRIORITY_FIRST_PROTOCOL' IN TYPEOF ( SELF\Environment. scheduler ) )
  R3 : SELF\Environment. scheduler. preemptivity <> partially_preemptive;
  R4 : SELF\Environment. scheduler. quantum = 0;
END_ENTITY;
END_SCHEMA;
```
Outline

1. Feasibility tests and real-time design patterns
2. Method from user’s point of view
3. Design Patterns Modeling
4. Feasibility Tests Selection Algorithm
5. Evaluation
6. Discussion
Feasibility Tests Selection Algorithm

Step 1  Model analysis to build dependency graph
Step 2  Graph analysis to extract potential design patterns instances
Step 3  Design pattern applicability constraints checking
Step 4  Composition Analysis
Step 5  Applicability constraints checking for tests selection
Case Study

- AADL model parsed by Cheddar
- Instanciated in Cheddar meta model
**Step 1 : Model analysis to build dependency Graph**

Built dependency graph

- One node for each task
- One edge for each dependency between two tasks
- One type of edge for each type of dependencies
- Graph built by analysis of system instance in Cheddar
Step 2 : Graph analysis to extract potential instances

- Formalisation of view upon dependency graph (by dependency type, connex components, processor, task type, etc)
- Each connex component with only one type of edge is a potential design pattern instance
Step 3: Design pattern applicability constraints checking

Design pattern constraints

- **R1** All threads are periodic
- **R5'** No buffer
- **R5''** No data component
- **R6** Data sharing protocol is sampled, immediate or delayed timing
- **R7** No hierarchical scheduler: no shared address spaces between processors

- For each potential instance:
  - All applicability constraints of the concerned design pattern are checked
  - If all applicability constraints are respected, we have a design pattern instance
Step 4: Composition analysis

Composition rules

- Unpl. ∪ Rav. ↦ Rav.
- Rav. ∪ Rav. ↦ Rav.

- Design pattern composition analyse to determine one system wide design pattern
- Work in progress, resolved for the three design patterns in current evaluation
- Identification of dominant design patterns based on feasibility tests study
Step 5: Applicability constraints checking for feasibility tests selection

- For each design pattern, we have defined a set of cases for feasibility tests selection
- Applicability constraints for each case are evaluated
- Selection of feasibility tests corresponding to respected applicability constraints
Outline

1. Feasibility tests and real-time design patterns
2. Method from user’s point of view
3. Design Patterns Modeling
4. Feasibility Tests Selection Algorithm
5. Evaluation
6. Discussion
Evaluation

Cheddar Engineering Process

- Prototype implemented manually and integrated to Cheddar
- Meta model elaboration and extension within Platypus
- The aim is to be able to generate the same prototype, based on the meta model
- Then we will be able to extend the number of design patterns at the meta level and generate automatically the functional selection tool
Conclusion

1. Approach enabling an automatic selection of feasibility tests with the use of AADL design patterns
2. Method from user’s point of view
3. Prototype available at: beru.univ-brest.fr/svn/CHEDDAR-2.0/

Ongoing works

1. More complex design pattern composition
2. Protocol for adding a new design pattern to the tool
3. Metric definition
4. New patterns, environments, feasibility tests, anti-patterns, etc.
Conclusion

1. Approach enabling an automatic selection of feasibility tests with the use of AADL design patterns
2. Method from user’s point of view
3. Prototype available at: beru.univ-brest.fr/svn/CHEDDAR-2.0/

Ongoing works

1. More complex design pattern composition
2. Protocol for adding a new design pattern to the tool
3. Metric definition
4. New patterns, environments, feasibility tests, anti-patterns, etc.

ACKNOWLEDGMENTS

We would like to thank Ellidiss Technologies and Region Bretagne for their support to this project.