UML-AADL’09: Towards a Model-Driven Approach for Mapping Requirements on AADL

Mathieu DELEHAYE
Christophe PONSARD

www.cetic.be
Your connection to ICT research
1. Motivation
2. NFR and tools survey
3. Goal-oriented Modelling
4. Requirements model mapping to AADL
5. Conclusions and perspectives

Running example: cruise control
1. **Motivation**
   2. NFR and tool survey
   3. Goal-oriented Modelling
   4. Requirements model mapping to AADL
   5. Conclusions and perspectives
Motivation: Bridging the Requirements to Architecture GAP

• Architectural design
  • Some styles, patterns
  • Remains much of an “art”

• Towards a model-driven approach
  • Architectural models, like AADL
  • Requirement models, goal-oriented approach, like KAOS
Functional Requirements vs Non-Functional Requirements

- **Functional Requirements (FR)**
  - define the functional effects the software-to-be is required to have on its environment
  - “WHAT” aspects
- **Non-Functional Requirements (NFR)**
  - define constraints on the way the software-to-be should satisfy its FR or on the way it should be developed
  - drive design choices
- In real-time embedded systems, NFR are as much important as FR!
1. Motivation

2. **NFR and tool survey**

3. Goal-oriented Modelling

4. Requirements model mapping to AADL

5. Conclusions and perspectives
Non-Functional Requirements for Embedded Systems

- Dependability requirements
  - RAMS, MTBF/MTTR/SIL
  - Formal verification
  - TOOL: TINA
- Performance requirements
  - Deadlock/starvation
  - Real-time constraints
  - Power consumption, mem footprint
  - TOOLS: CAT, Cheddar, PathCrawler
- Security
  - Confidentiality, security, availability
Outline

1. Motivation
2. NFR and tool survey
3. **Goal-oriented Modelling**
4. Requirements model mapping to AADL
5. Conclusions and perspectives
Goal-Oriented Requirements Engineering
Goal Model of the Cruise Control System

- Maintain [Speed Safely]
- Maintain [Speed When Engaged]
- Maintain [Braking Allowed All Time]
- Maintain [Driver In Control]
- Maintain [CC Controlled]
- Motor behavior
- Braking WHEN CC engaged
- Achieve [Braking disengaging CC]
- Control Speed
- CC System
- Press Brake
- Performance
- Disengage CC
- Engage CC
- Set Speed
- Driver
Formal level: goals

**Requirement** Maintain[SpeedControlledWHENEngaged]

**FormalDef** \( cc \cdot engaged \)

\[\Rightarrow | \text{car.speed} - \text{cc.targetSpeed} | < \text{cc.margin} \]

**Requirement** Achieve[CruiseDisabledWHENBraking]

**FormalDef** \( \text{brakePedal.pressed} \)

\[\Rightarrow \lozenge_{\text{max}} \neg \text{cruiseControl.state=ENABLED} \]
Formal level: operation

**Operation**  ComputeTorque

**Resp**  CruiseController

**Input**  \(car\text{.}speed, cc\text{.}targetSpeed\)

**Output**  \(car\text{.}torque\)

**ReqPre**  for Achieve\([\text{CruiseDisabled\ WHEN\ Braking}]\) :

\[ cc\text{.}engaged \]

**ReqPost**  for Maintain\([\text{Torque\ Computed}]\) :

\[ car\text{.}torque = f(car\text{.}speed, cc\text{.}targetSpeed) \]
Agent Model: a High Level Architecture

- Outcome of the goal-oriented process
- Information flow between agent
- Based on responsibilities (goals) and capabilities to monitor/control information (operation)
Outline

1. Motivation
2. NFR and tool survey
3. Goal-oriented Modelling
4. Requirements model mapping to AADL
5. Conclusions and perspectives
Quick reminder on (A)ADL

- **Components:** HW or SW, 2 levels of description
- **Connectors:** data and control flows

**Execution platform components**

- Processor
- Memory
- Device
- Bus

**Application software components**

- Process
- Subprogram
- Threadgroup
- Thread
- Data

**Composite components**

- System
Principle of Mapping Requirements and AADL Models

**Requirements Model**

- **CruiseControl**
  - Goal Model
  - Object Model
    - Car
    - Brake
    - Motor
  - Agent Model
    - Maintain[Speed]

**AADL Model**

- AADL - structural part
- AADL - behavioral part

```plaintext
thread implementation CC.impl
  annex behavior_specification
  {*
    states
    torque: Behavior::Integer;
    transitions
    compute -> [new Torque?(torque)
      when cc.engaged]
    -> compute};
  **}
end AP.impl;
```
Global mapping process

- producing the AADL structural description
  - mostly based on the agent model,
  - complemented by the goal and object models.
- producing the AADL behavioral part,
  - based on the operation model
- refining the initial architecture
  - by injecting NFR based on a pattern library.
Deriving the structural part

- each of this software process corresponds to an agent of the requirements model.
- for each of those agent, the inner operations under its operation are depicted as components with:
  - a data input port defined for each input of the operation
  - a data output port defined for each input of the operation
  - a control port is defined for each variable of the trigger part of the operation
- additionally, use AADL properties for specific requirements (e.g. temporal constraints as deadlines)
Generating the behavioural part

1. Generation of Finite State Machine (FSM) by a goal based generator
   • Input: a collection of operations under the responsibility of an agent
   • Output: FSM describing the states and transitions logic expressing the objects and operations.

2. Generation of AADL mode-transition based on the generated FSM. The structure of the FSM is recoded in AADL style.
Generating the behavioural part

StartCC
DomPre: CC.OFF
DomPost: CC.ON

StopCC
DomPre: CC.ON
DomPost: CC.OFF

SetSpeed
DomPre: CC.DIS
DomPost: CC.ENG

Resume
DomPre: CC.DIS
DomPost: CC.ENG
ReqPre: not brake.pressed

Brake
DomPre: not braking
DomPost: braking
ReqTrig: brake.pressed

CC controller
OFF
stopCC

startCC

ON

DISengaged

setSpeed

Resume
[not brake.pressed]

ENGaged

brake.pressed
Typical Behavioral Description of the Cruise Control in AADL

thread CruiseController
features
  speed: in data port;
  brake: in data port;
  ccPressed: in event port;
  resumePressed: in event port;
  torque: out data port;
end CruiseController;

thread implementation CruiseController.i

annex behaviorspecification{**
states
  OFF: initial states;
  ON: complete join state;

transitions
  OFF-[ccPressed?] -> ON;
  ON-[ccPressed?] -> OFF;
Composition state ON
  states
    DIS: initial state;
    ENG: complete state;
  transitions
    DIS-[resume?] -> ENG;
    ENG-[on brake>0] -> DIS;
    ENG-[on |speed-targetSpeed|>margin] -> ENG
{torque=fn(speed,targetSpeed)};
end ON ;
**};
end CruiseController
NFR-driven refinement (on-going)

- Refining architecture by adding/replacing/constraining components and connectors
- Based on pattern approach (reusable)
  - Category specific patterns: availability, accuracy, fault tolerance, interoperability, security
  - Domain specific patterns (e.g. embedded systems): Actuator-Sensor, Controller Decompose (splitting responsibilities), Monitor-Actuator (reliability), Fault Handler (fault tolerance), Watchdog (reliability)
Example: data integrity connector

NFR: Maintain[DataIntegrity]

Error
Outline

1. Motivation
2. NFR and tool survey
3. Goal-oriented Modelling
4. Requirements model mapping to AADL
5. Conclusions and perspectives
Conclusions

• Model-driven method to derive initial architecture from requirements models

• Benefits of requirement models:
  • early reasoning on FR (completeness, consistency)
  • identification of NFR and connection with FR

• NFR taken into account in this process, also taking into account specificities of embedded systems, allowing the analyst to stay in control of the way he wants to transition from the problem to the solution space.

• This work was restricted to the context of goal-based requirements and AADL model but can be generalized.
Future Work

• More research on architecture refinement driven by NFR, especially related to embedded systems.

• Implement a prototype of the mapping based on the Objectiver and TOPCASED tools.

• Encode and enrich pattern library.

• Moreover the alignment of the requirements approach with existing standards such as UML/MARTE or SysML must also be considered.
Questions ?