Using the Model Paradigm for Real-Time Systems Development: ACCORD/UML

OOIS’2002 – Montpellier, September the 2nd
Workshop on Model-Driven Approaches to Software Development

Dr. Sébastien Gérard
CEA – LIST / L-LSP
Sebastien.Gerard@cea.fr
Plan of the presentation

- What is MDA?
- A full Model-Driven Methodology for ESD
  - ACCORD/UML
- Examples of MDE implementation
- Wealth of MDA technologies …
What is MDA?

1 - Model Driven Architecture means
   • Model → UML models
   • Architecture → Engineering!
   → Modelling with UML to develop a system …

2 - Model Driven Architecture involves
   • PIM
   • PSM
   … with a clear implementation technology separation …

3 - Model Driven Architecture relates to
   • Model transformation
   … following a continuous modelling process and using model intensively synthesis.

MDA means modelling with UML to develop a system with a clear implementation technology separation following a continuous modelling process and using model intensively synthesis.
Outlines of the ACCORD/UML methodology

- **Result of**
  - + 10 years of OO researches for RT-Syst. for nuclear plant
  - PhD work with PSA (Peugeot)
  - AIT-WOODDES, an IST European project
Models everywhere …

Application synthesis

Design Mappings

Prototype Model

Code Mappings

Application Code

Compiling & Link Mappings

Executable Application

Analysis Model

UML2AGATHA Mappings

Formal Language of AGATHA

SET2UML Mapping

Symbolic Execution Tree

Test Model

Test Synthesis

Application synthesis
Usual way to use the state machine …

Regulator

tgSpeed : int

initReg()
stopReg()

Logic & Algorithmic

Maintainability ▼

Reusability ▼

Logic

Algorithmic

initReg[cptVit->getSpeed()=<30]
/display("ON");

stopReg/display("OFF");

[carSpeed=<30]/display("OFF");

tgSpeed = cptVit->getSpeed();

delta=k1*atan(tgSpeed-cuSpeed);

tm(100)/tgSpeed = cptVit->getSpeed();

mot->sendCmd(coupleVariation);

carSpeed=<30]/display("OFF");
Separation on concerns to improve behaviour modelling

Two aspects = Logic & Algorithmic
Protocol state-machines and activity diagrams

Class behavior =
Control logic (protocol use)

Method behavior
Algorithmic parts

Protocol-transition
call-event (’params’) ’[’ guard ‘]’

Maintainability ➔
Reusability ➔

Regulator interface
« describe reactive view »

Class interface

Regulator
+tgSpeed : integer
+initReg()
+stopReg()
+maintainSp()

« Signal »
RegOnOff()

Triggering view (ACCORD/UML)

RegOnOff / initReg()
\ / maintainSp()
On
RegOnOff / stopReg()
Off
[carSpeed<50] / stopReg()

ChangeEvent
CompletionEvent
SignalEvent
TimeEvent

Triggering-transition

`evt-name ' ('param-list')' [ 'guard ' ] '/ <ope-name> ' ('params')'

Set of consistent rules between protocol and triggering views
Démonstration de l’approche ACCORD/UML

Démarrer la démonstration ...

Example of a synchronization management: classical view

1. **Data declaration:**
   ```
   ...  
   double tgSpeed;
   ```

2. **Data protection creation:**
   ```
   ...  
   SEM_ID semTgSpeed;  
   semTgSpeed = semMCreate(...);
   ```

3. **Writing access:**
   ```
   ...  
   void maintain ()
   {
   semTake (semTgSpeed, WAIT_FOREVER );
   if ( carSpeed > tgSpeed ) ...  
   semGive (semTgSpeed ); }
   ```

4. **Task creation:**
   ```
   ...  
   taskSpawn("T1", priority, ..., maintain, ...);
   ...  
   void maintain () { ... }
   ```

- This approach needs low level code integration
- The programmer have to forget nothing …
- The user have to know the task implementing
Example of a synchronization management: ACCORD view

1. **Data declaration and protection creation:**
   - Encapsulated in a class

2. **Shared data access:**
   - The owning class have direct access
   - Else, access via a special operation
     ```java
     aReg.get_tgSpeed();
     ```

3. **Task creation:**
   - To declare the object as task server
   - To add operations to the object
     ```java
     void maintain() { if (carSpeed > get_tgSpeed()) ... }
     ```

- **Automatic implementing (transparent to developer)**
- **Only need declaration of constraints to respect**

```
Regulator « RealTimeObject »

double get_tgSpeed();
maintain();
double tgSpeed;
```
Automatic synthesis of component model

ER1 = Heuristics to ease decomposition of a system into a sub-systems model

ER2 = Model mappings to transform resulting sub-systems model into component model
- required & offered interface
- component connexions / association links
- ...

Component definition
→ 2 possible views:
- “Black box” (= external view)  ➔ set of interfaces
- “White box” (= internal view) ➔ sub-system
Automatic synthesis of component model (con’t)

ER2 = These mappings take into account specified RT features of the sub-system model

→ PIM-to-PIM

ER3 = Mappings definition from the RT-Component Model of ACCORD/UML towards a platform model (e.g., RT-CCM model)

→ PIM-to-PSM
The designer invokes our tool which translates UML models into an EIOLTS specification.

- AGATHA generates an exhaustive functional tree with associated Path Conditions which is converted into UML Sequence Diagrams.
- The designer can study these diagrams and fix the errors in the model.
- This process can be repeated until the model is correct.

⇒ Fully transparent processus: the user does not have to know anything about the “formal” language involved in validation process.
Wealth of MDA technologies …

- **MOF- vs. Profile-based approaches**
- **Separation of Concerns**
  - Aspect Oriented Modelling vs. Meta-Objects Modelling vs. Pattern Oriented Modelling vs. …
- **Model transformation language**
  - what are the real issues ???
  - no graphical proposals based on UML ???
- **exTreming Modelling, Adaptative Programming, AOM, MOP, Code generation, … "Help, I am lost!"**
Conclusions

- **NEEDS of a Technologies Map**
  - listing of possible support technologies
  - describing when, how, for what a technology is efficient/useful regarding other one
  
  ➔ to clarify the situation

- **Modelling is not new !**
  - Moderation around the new key-word MDA

- **UML 2 ... ☺ VS. ☹**