Multi toolset Integration:
AADL, ASN.1, SDL, SCADE, STOOD and Ocarina

Applying Model-Driven Engineering Concepts to build High-Integrity systems in the IST-ASSERT process

Jérôme Hugues, ENST (speaker)

With the help from
• Audrey Shaufelberger, ESA
• Dimitrios Mylonas, ESA
• Maxime Perrotin, ESA
• Cyril Colombo, ESA
• David Lesens, AST-F
• Pierre Dissaux, Ellidiss
• Thanassis Tsiodras, Semantix
ASSERT context: building complex systems correctly

- ASSERT is an FP6 IP project, ended Jan 2008
  - 30+ partners, 3 years
- Desired functionality – and coding complexity - have reached levels (dangerously) close to being unmanageable by humans. “Empirical” methodologies have reached their limit...
- Model Driven Engineering offers formal verifications of correctness, at the model level
- Use of appropriate modeling language per domain
- Different tools generate different code – no established standard for interworking between generated code
- Messages/data exchanged between different tools face the same issue – no common data modeling
- Must be completely automated – nothing should be left to manual labor
MSU model overview

Cyclic @ 10Hz

Basic Operations

Thrusters activation

Activates every 500 ms

Upstream

Basic block

End boost reached, sun is aimed

Downstream

Control block

Control-logic and Control-laws FV: use of data-flow oriented languages by re-using existing MSU Scade model as building blocks

Requires further modeling of the behavior of the system with FSMs: use SDL to create the behavioral model

HLTC, 
Nav acq.
FTCP Health Sts

HLTM
Overview of the ASSERT process

Tool chain allowing the user to capture his system using a set of models and to generate the complete code of the application without manual intervention
The ASSERT process

System Model

Data Models

Lustre SDL

ObjectGeode

SCADE

The source code for each subsystem as created in Stood (and asn2aadlPlus)

The semantically equivalent types of the messages per tool

The messages exchanged between subsystems

The encoders and decoders of the ASN.1 messages

The code generators of the source code

The编译器

The C code

Behavior

Data structures

aadl

AADL Container

AADL Runtime

System Model

Data Models

Lustre SDL

ObjectGeode

SCADE

The source code for each subsystem as created in Stood (and asn2aadlPlus)

The semantically equivalent types of the messages per tool

The messages exchanged between subsystems

The encoders and decoders of the ASN.1 messages

The code generators of the source code

The ASN1 compiler

(Asn1c)

C code

Behavior

Data structures

aadl

AADL Container

AADL Runtime
ASSERT process work breakdown

- **Data View (ASN.1)**
  - Define the ASN.1 types

- **Interface View, using STOOD (AADLv1 + extensions)**
  - Import Dataview
  - Create the interface view, reference the functional view

- **Functional Views (C, Ada, SDL, SCADE, Simulink ...)**
  - **SDL**: import DataView, model the behavior of the system with state machines & generate code
  - **Scade**: import DataView, import Interface View, plug existing SCADE blocks & generate code

- **Deployment/Concurrency View, using plain AADLv1**
  - Capture physical architecture of the system
  - **Automatically generated** from the other views

- **Group, compile & link together...and execute !**
The Data View

- Using ASN.1 (ITU-T standard)
- Unique “neutral” data definition at system level
- Ensures the global consistency of data types shared between functional models

T-MATRIX-3-4 ::= SEQUENCE (SIZE(3)) OF T-MATRIX-3-4-LINE

T-MSU-ID ::= ENUMERATED {
  msu1 (0),
  msu2 (1)
}

T-MSU-STATUS ::= ENUMERATED {
  slave(0),
  master(1)
}

T-MSU-TELEMETRY ::= SEQUENCE {
  state-is-pfs-m-stdby BOOLEAN,
  state-is-pfs-m-ready-for-cam BOOLEAN,
  state-is-pfs-m-cam-in-waiting BOOLEAN,
  state-is-pfs-m-cam-prgs BOOLEAN,
  state-is-pfs-m-cam-inhib BOOLEAN,
  msu-failed-has-been-detected BOOLEAN,
  cam-mode T-CAM-MODE,
  ...
}
The interface view: APLC definition, PI and RI creation
http://www.ellidiss.com

- Modeling with STOOD by Ellidiss
  - Use of AADL systems to structure the system
  - PI are event ports triggering subprograms
  - RI are out event ports associated to remote calls
  - Support of Ravenscar semantics: AADL properties set
The interface view: linking to PI and RI to functional views

Set of Subprograms defining a functional view

Subprogram featuring:
- Implementation language
- In and out parameters imported from ASN1 model
- Out event ports for remote procedure calls

Invocation of a Scade function

Invocation of a SDL function
The SDL behavioral Model (Example of the Basic FSM)

Definition of the SDL signals

Definition of the Scade function call

ASN1 imported type
The SDL behavioral Model (Example of the Basic FSM)

**Reception of the SDL signals**
- **Cyclic Activation Implementation**
  - **writein** (**Cyclic Activation***)
    - \( i = i + 1 \)
    - \( j = j + 1 \)
    - **writein** (**C**)
    - **writein** (**I**)

**Updating internals on reception of data issued from US and DS**
- **Control UP TO Basic (ctrlUPOUT)**
  - **writein** (**RECEIVE ControlUPTOBasic***)
    - **writein** (**thrusters_opening = ctrlUPOUTthrusters_opening**)
- **Control DOWN TO Basic (ctrlDOWNOUT)**
  - **writein** (**RECEIVE ControlDOWNTOBasic***)
    - **end_boost_is_reached = ctrlDOWNOUTend_boost_is_reached, sun_is_arrested = ctrlDOWNOUTsun_is_arrested**
    - **writein** (**thrusters_opening**)
    - **writein** (**end_boost_is_reached")
    - **writein** (**sun_is_arrested")

**Simulation**
- (true)
  - Simulation
- (false)
  - Simulation
Scade functional view

Re-utilization of an existing model from EADS case study as **functional** building Blocks for our demonstrator
AADL “Tetris”

- Stood maps from APLC to VMLC (concurrency view)
- VMLC-view is AADLv1 view of the system
  - The envelope of the application: threads, buses, ...
  - Additional “glue” code to interconnect components: ASN.1 marshallers, stub/skel for interactions
- Each component plays a particular role
  - Described in the most suited representation (ASN.1, SCADE, SDL, Simulink, Ada or C source code...)
  - Then abstracted as an AADL component
- AADL serves as a common representation to describe each components
  - AADL Component type to reflect its category
    - System, processor, data, etc
  - AADL Properties to configure it
    - Period, priority, WCET, resource consumption, etc
- AADL connects all components
- The construction of an application following the AADL process is a precise game of “Tetris”
AADL “Tetris”: from VMLC to code

- Take the concurrency view, map it onto Ada code
  - Compliant with the Ravenscar profile (provable concurrency)
  - Compliant with all HI guidelines set by ESA
  - Support for distribution
  - Baselined on AADLv1

- Rationale: for each AADL entity, a Ravenscar-compliant pattern has been defined. Code generator maps them correctly w.r.t VMLC model topology

- Incorporate ASN.1 wrappers to address heterogeneity in the model, and ensure interoperability

- Ocarina: AADL-to-X compiler
  - PolyORB-HI: runtime for Ocarina, support ASSERT runtime requirements for HRT, distribution using gnatforleon (UPM) and SpaceWire, MTS (SciSys)

- ASN.1 tools by Semantix
  - Address interoperability between all functional code (e.g. interaction between SCADE, SDL and Ada code)
AADL “Tetris”: Data component

- **asn2AADL (Semantix)**: make types visible at AADL level, to be exploited by concurrency view
- **aadl2glueC**: ASN.1 marshallers
  - Builds Ada+C functions to convert data to/from ASN.1,
    - Exports them as AADL subprograms
    - Used to pass data from one APLC to another to ensure interoperability between SDL/SCADE/Ada/C/..

```plaintext
DATA T_MSU_TELEMETRY
PROPERTIES
  Source_Text => ("DataView.asn");
  Source_Language => ASN1;
  Source_Data_Size => 1024 B;
  Type_Source_Name => "T-MSU-TELEMETRY";
END T_MSU_TELEMETRY;

DATA T_MSU_TELEMETRY_Buffer END T_MSU_TELEMETRY_Buffer;

DATA IMPLEMENTATION T_MSU_TELEMETRY_Buffer.impl
  -- Buffer to hold a marshalled data
  SUBCOMPONENTS
    values: data ASSERT_Types::Stream_Element;
  PROPERTIES
    ARAO::Length => 1024; -- Size of the buffer
END T_POS_Buffer.impl;
```
AADL “Tetris”: concurrency entities (example)

- Reusable, provable pattern for each concurrency entity (periodic, sporadic, protected, ...)
- Ex: cyclic container

```plaintext
thread BASIC_APLC_Cyclic_FV
features
  CyclicActivationImplementation : out event port;
end BASIC_APLC_Cyclic_FV;

thread implementation BASIC_APLC_Cyclic_FV.Impl
calls { Wrapper : subprogram BASIC_APLC_Cyclic_FV(SDL_ADA); }
properties
  Dispatch_Protocol => Periodic;
  Period => 1000 ms;
  Deadline => 1000 ms;
  ARAO::Priority => 2;
end GNC_thread.Impl;

package BASIC_APLC_Cyclic_FV_Task is
new PolyORB_HI.Periodic_Task
(Entity => BASIC_APLC_Cyclic_FV_K,
Task_Period => Milliseconds (1000),
Task_Priority => 2,
Task_Stack_Size => 64000,
Job => BASIC_APLC_Cyclic_FV(SDL_ADA));
```
AADL “Tetris”: summary 1/2

Round#1: from APLC to VMLC

1. Model your system as a set of APLCs
   - Use ASSERT AADL extensions to model components
   - Interconnect them
   - Reference SDL/SCADE/C/Ada models

2. Check model semantics

3. Generate VMLCs
   - Define the envelope of the application
AADL “Tetris”: summary 2/2

- Round#2: from VMLC to application

1. Generate AADL view of VMLC
2. Generate AADL view of ASN.1 data types
3. Generate AADL + Ada/C code for ASN.1 marshallers
4. Integrate functional code
   - SCADE’s C code, SDL’s C code, Ada, C ...
5. Generate Ada code for AADL VMCLs
6. Configure AADL runtime
7. Link altogether
   - Plus Ada runtime, SpaceWire drivers
8. Test your application!
### Execution outputs

<table>
<thead>
<tr>
<th>INPUT</th>
<th>OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>pfc</td>
</tr>
<tr>
<td>5</td>
<td>new configuration available: true</td>
</tr>
<tr>
<td>10</td>
<td>new configuration available: false</td>
</tr>
<tr>
<td>15</td>
<td>configuration is on: true</td>
</tr>
<tr>
<td>20</td>
<td>configuration is on: false</td>
</tr>
<tr>
<td>25</td>
<td>arming relay status on: true</td>
</tr>
<tr>
<td>30</td>
<td>red button is on: true</td>
</tr>
<tr>
<td>35</td>
<td>CAM trigger status: true</td>
</tr>
</tbody>
</table>

CAM in progress: true
Ready for CAM: false
What as been experienced in ASSERT demonstrator

- **Transparent use** of complementary languages (SCADE and SDL). Data types are converted automatically from one language to the other (both at model and code level)
- **Automatic generation** of binary encoders and decoders for data exchanged in distributed systems
- **Use of AADL** for a precise description of the system architecture and properties: no ambiguity, easy verification and code generation with existing tools
- Use of **SDL** for the description of the system functional behavior: appeared to be precise and very easy to use.
- Use of **SCADE** for the description of the system algorithms: dedicated, appropriate language (but not appropriate for state machines)
- **AADL as a backbone to integrate** all concerns seamlessly, using STOOD & Ocarina
assert project
For a reliable and scientific approach in system and software engineering
http://www.assert-project.net