Leveraging AADL for High Level Architecture Modeling, Analysis and Generation

The ASSERT Approach
Starting with a statement of the problem.

- Targeted domain: Real-Time, mission critical and embedded system.
- SW dominant systems (But expandable to HW)
- Current development approach: mostly based on paper work.
  - Few models,
  - Tools: mainly MS office!
- Suppliers face difficulties to master design before testing
- Customers find reviews not efficient enough.
- Needs:
  - Capture system model with associated properties,
  - Verify early and continuously during design,
  - Smoothly handle system heterogeneity,
  - Use automatic coding.

**Those needs were addressed in ASSERT.**
The ASSERT Requirement baseline

- System families: from market segments to property oriented design,
- Proofs: from an empirical to a scientific approach,
- New development process: from nice concepts to actual steps,
- Tools: From paper to models,
- Heterogeneity handling: from multiple models to one single and integrated SW.
- Case studies: from toy examples to real cases.

**ASSERT = an ambitious and pragmatic approach to develop critical SW dominant RT systems.**
Bright and dark faces of our achievements.

- **System families:**
  - No usable implementation of reference architectures,
  - Technically very complex,
  - New business model to be setup,
  - Requirement capture phase too much driven towards existing solutions.

- **The bright face: a process and a toolset.**
  - Process is covering all phases from SW system definition and properties capture to SW generation and deployment,
  - The toolset fully supports all process steps and uses AADL as a system definition language.
Why has AADL been chosen?

• The ASSERT process is independant of any technology,

• Two different toolsets have been initiated in ASSERT:
  – One supports the AADL language (the one we are showing in this presentation).
  – A second one is using HRT-UML.

• The choice of AADL was motivated by:
  – The maturity and adequacy of the language,
  – A simple and readable textual formalism,
  – The facilities for capturing system properties,
  – The ability of making connections to external languages
  – The existence of the growing community around the AADL committee.
ASSERT and post-ASSERT activities

- The existing results we will show have been produced by both ASSERT and follow-up activities.
- The ASSERT IP delivered:
  - A first definition of the process,
  - Some early prototypes of tools,
  - First case studies,
- ASSERT follow-up activities:
  - Were funded by ESA (internal R&D Budget)
  - Transformed the first tool versions into a workable toolset,
  - Complete the list of case studies with additional examples.
Future of ASSERT: a strategy around three main axis.

- **Extending the process and toolset**
  - Link with system modelling: an ESA funded study to be kicked-off very soon.
  - Integration of HW components: Another ESA funded study to be started.
  - Integration of space specific components (PUS, Spacewire, ...): partly done but to be completed.
  - Connection to system simulator and schedulability analyser.

- **Market the toolset**
  - Create an ecosystem gathering innovative SMEs and research centres into a network,
  - Each network node will lead a technical domain,
  - The network can be dynamically configured to address a specific need (virtual company),
  - To have ESA coordinating the network and protecting the community investment.

- **Disseminate the technology.**
  - Case study on formation flying experiment with space industry (Spacebel and Space System Finland).
  - Preparation of case studies with space and non space industry.
Back to your future ... toolset!
What is the Assert process?

- **The Assert process is based on simple observations**
  - a system – ANY system – is made of *heterogeneous components*, that have to live and communicate together
  - system builders have other concerns than *software implementation details*,
  - and good software engineers are unhappy when they have to develop application code: their skills are misused

- **The Assert process proposes solutions to**
  - capture a system using user-friendly (yet formal) modelling techniques
  - automate repetitive and error-prone software activities
  - build an homogeneous system having heterogeneous components

- **The toolset has been specified, designed, and implemented by ESA together with some Assert partners.**

- **It is unique on the market.**
Capture of the system: architecture, behaviour, data, real-time attributes, and hardware platform.

**AADL and ASN.1** are combined to provide a formal, precise, and complete description of the system architecture and data.
What Assert tools do with the models

1. Generate “application skeletons” in Simulink, SDL, C, and Ada

2. Generate a software real-time architecture (in AADL)

3. Generate glue code to put everything together on a real-time operating system
In addition - bonuses

- **Rapid prototyping:** the toolchain generates GUIs to quickly test the system under development

- **Simulation and Analysis:** Data can be monitored using real-time plotting.

- **Documentation:** ICDs are generated automatically with a description of the data binary encodings (ASN.1 uPER Encodings)

```plaintext
MY-MODULE DEFINITIONS ::= BEGIN

MySequence ::= SEQUENCE {
  field1 INTEGER (5..4294967295),
  field2 INTEGER (5..4096) OPTIONAL,
  field3 BOOLEAN ,
  field4 MyChoice,
}
```

---

RST Ada Europe 2009 - 12/06/09

assert-project - Dr Eric Conquet & Maxime Perrotin – European Space Agency
A few screenshots of the toolset (1)

- Interface and deployment view editors
A few screenshots of the toolset (2)

- Generation of Simulink “skeletons”
A few screenshots of the toolset (3)

- **Generation of SDL “skeletons”**

```plaintext
system basic_fv

USE Datamodel;
SIGNAL basictotc (T_TM);
SIGNAL tcommand (T_HLTC_PLUS);
SIGNAL basictocontrol (T_CONTROL_IN);
SIGNAL controldowntobasic (T_CONTROL_DOWN_OUT);
SIGNAL controlduptobasic (T_CONTROL_UP_OUT);
SIGNAL cyclicactivationimplementation;

procedure aplc_basic_op COMMENT "#c_predef":FPAR
  IN thrusters_opening T_THRUSTERS_OPENING,
  IN pfs_lw_m_ariming_relay_status_on T_PFS_LWM_ARMING_RELAY_STATUS_ON,
  IN pfs_hltc_red_button_is_on T_PFS_HLTC_RED_BUTTON_IS_ON,
  IN msu_id T_MSU_ID,
  IN pfs_ew_m_msuy_msux_hs T_PFS_EWM_MSU_MSU_HS,
  IN tcp_health_status T_FTCP_HEALTH_STATUS,
  IN pfs_ew_m_dtg12_msu T_PFS_EWM_DTG12_MSU,
  IN hltc T_HLTC,
  IN end_boost_is_reached T_END_BOOST_IS_REACHED,
  IN sun_is_aimed T_SUN_IS_AIMED,
  INOUT pfs_ewc_msu_pde_t T_PFS_EWC_MSU_PDE_T,
  INOUT pde_cmd_a T_PDE_CMD_A,
  INOUT dpu_cmd T_DPU_CMD,
  INOUT set_pfs_ewc_msu_dtg_mode_coarse T_ON_OFF_CMD,
  INOUT hltm T_HLTM,
  INOUT pfs_ew_m_msuy_msux_hs T_PFS_EWM_MSU_MSU_HS,
  INOUT cam_mode T_CAM_MODE,
  INOUT controller_to_be_activated T_CONTROLLER_TO_BE_ACTIVATED,
  INOUT navigation_output T_NAVIGATION_OUTPUT;
EXTERNAL:

procedure mysimulink COMMENT "#c_predef":FPAR
  IN my_in T_FOR_SIMULINK_IN,
  IN my_in2 T_control_in,
  INOUT my_out T_FOR_SIMULINK_OUT,
  INOUT my_out2 T_Control_in;
EXTERNAL:
```

---

assertation

assert project

assert assertion

assert project
A few screenshots of the toolset (4)

• **C “skeletons”**

```c
#include "user_code.h"

void cyclicactivationimplfortc()
{
    /* Write your code here */
}
```

```c
/* This file was generated automatically: DO NOT MODIFY IT */

#ifdef __USER_CODE_H_tc
#define __USER_CODE_H_tc
#endif

#include "C_ASM1_Types.h"

extern void tcommand(const T_HLT_CPLUS *);

void cyclicactivationimplfortc();

#endif
```
A few screenshots of the toolset (5)

- Ada “skeletons”
The ASSERT Process in action
Prepared by Cyril Colombo and Marie-Aude Esteve (ESA)
Case study for this demonstration the ATV docking sequence

Arbitration logic must:
- respect reactivity constraints
- be robust to any nominal or non nominal situation
- be robust to asynchronous request sequence such as TC
How to integrate this behavioural model in our example using the ASSERT process?

Select an appropriate modeling language to model the expected behavior

SDL is a very good candidate in this situation

- Allows to clearly specify the expected behavior as a Finite State Machine
- Allows to further check some properties at model level i.e. prior to integration at code level
- Allows to be transparently integrated in the application thanks to the ASSERT tool chain

Simply defining the modeling language at Interface View level allows:

1. To automatically generate a correct by construction model interfaces & skeleton from where to start the modeling work

1. To regenerate automatically the full system without having to write manually a single line of code!
Examples of proofs made on the SDL model

• “The reception of the TC disabling the PCF ensures in any case the selection of the GNC as the command source”

• “The PCF state of the automaton will be left at least at the expiration of the PCF timer”

• “There is no combination of input leading to a deadlock of the automaton”

Results with exhaustive simulation:

- Number of states: 102008
- Number of transitions: 397474
- Maximum depth reached: 259
- Maximum breadth reached: 508
- Duration: 0 min 8 s
- Number of exceptions: 0
- Number of deadlocks: 0
- Number of stop conditions: 0
- Transitions coverage rate: 100.00% (0 transitions not covered)
- States coverage rate: 100.00% (0 states not covered)
- Basic blocks coverage rate: 100.00% (0 basic blocks not covered)
- Number of errors: 0
- Number of success: 102
Assert toolset - Credits

- Interface View
  - LabASSERT (Ellidiss)

- Model Transformation
  - Buildsupport (ESA)

- Data Modeling
  - ASN.1 tools (Semantix)

- Functional View
  - Ada, C
  - Matlab - Simulink
  - SDL

- Automatic Code generation
  - Ocarina (ENST)
  - Assert builder (Semantix, ESA)

- Virtual Machine
  - PolyORB-HI (ENST)
  - GnatForLeon (UPM)