

# AADL model transformations

P. Dissaux\*

*\*TNI-Europe & Ellidiss Technologies,  
Technopôle Brest-Iroise, 115 rue Claude Chappe,  
F-29280 Plouzané, France  
[pierre.dissaux@tni-world.com](mailto:pierre.dissaux@tni-world.com)  
<http://www.tni-world.com>*

## Abstract

The Architectural Analysis and Design Language (AADL) is an Architecture Description Language (ADL) that has been standardized in November 2004 by the Society of Automotive Engineers (SAE), Aerospace Avionics Systems Division. The scope of this language is quite wide. It targets critical developments and offers appropriate modelling techniques to cover system and software engineering activities.

Taking a Software Engineering point of view, this paper firstly presents a few possible benefits of using the AADL within a critical software development process, then describes the model transformations plugins that have been implemented into the Stood 5.0 software design tool to support AADL 1.0 in connection with UML 2.0 graphical notation while preserving the benefit of HOOD top-down modelling process and HRT-HOOD real-time semantics.

## 1 Using the AADL in a Software development process

Let's consider some of the main concerns that arise during Software Engineering activities:

- How to import System Engineering architectures as blueprints for the Software Architectural Design activities?
- How to refine System Engineering Components with the output of the Software Design activities?
- How to build a repository of reusable Components from legacy source code?

The use of a language that can carry the common application architecture throughout the development lifecycle, while accepting the customized detailed that are needed by each modelling step, is required to reach these goals. In addition, such a language must be able to efficiently carry advanced concepts like real time paradigms.

Such a common language could be simply UML or XML, but the former is often perceived as being too "software oriented" for System Engineering purposes, which seems confirmed by the current OMG effort to define SysML, and the latter is only standardized at a syntactic level and thus requires the definition of a dedicated DTD or Schema, which raises an issue while considering it as a really recognized international standard.

The AADL offers the advantage of actually being an international standard, while precisely defining the semantics of a predefined set of Components, including real time concerns. It may also be handled either through its specific textual notation or through a XML syntax. The definition of the AADL as a UML profile is also on progress, in collaboration with the OMG.

Below are described the AADL-based solutions that have been implemented into the Stood 5.0 tool in order to cover the three concerns that have been identified above.

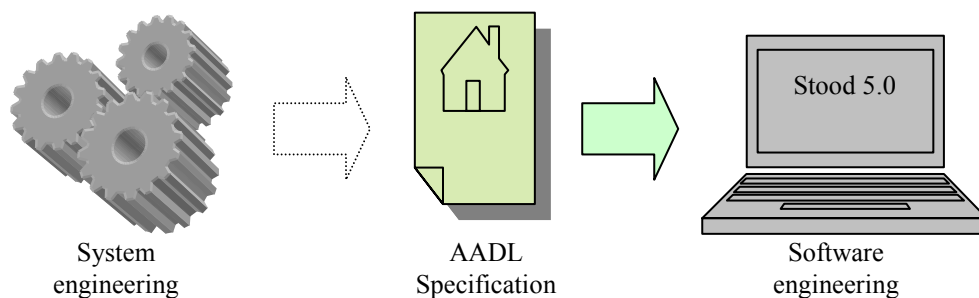
## 1.1 Bridging System and Software architectures

In that context, it is assumed that high level architecture of the System is described in such a way that it is possible to produce the corresponding AADL specification. This can be achieved either by editing directly the AADL code with a dedicated editor, or better, by adding AADL code generation features to existing System engineering modelling tools.

The role of the AADL specification is there to propagate the initial architectural choices, together with the right glossary and the specialized properties that should remain invariant all along the life cycle.

A Software design tool must then be able to import this architectural specification to automatically initialize a set of corresponding Software components that will become the initial draft for the purpose of the Software Architectural Design phase.

To reach this goal, the Software design tool must implement an AADL source code import feature. This feature must efficiently and automatically transform AADL constructs into Software design constructs.

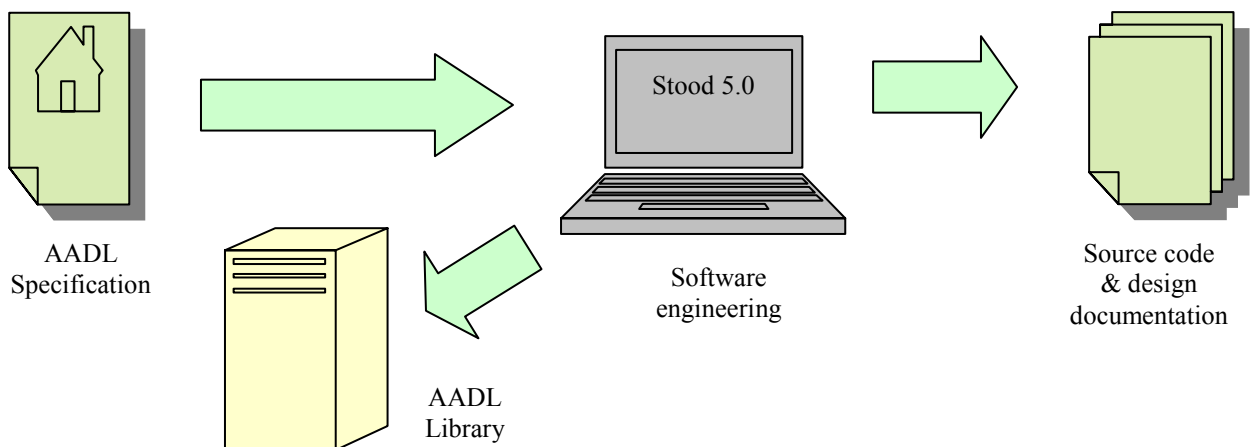


## 1.2 Refining reusable components with Software design items

Considering now the case where high-level components have been specified during System engineering phase, and imported into the Software design tool for Software engineering activities. The initial components will then be refined by the addition of a hierarchy of subcomponents as well as lower level properties that will fulfil the Software requirements while preserving the global architecture of the system.

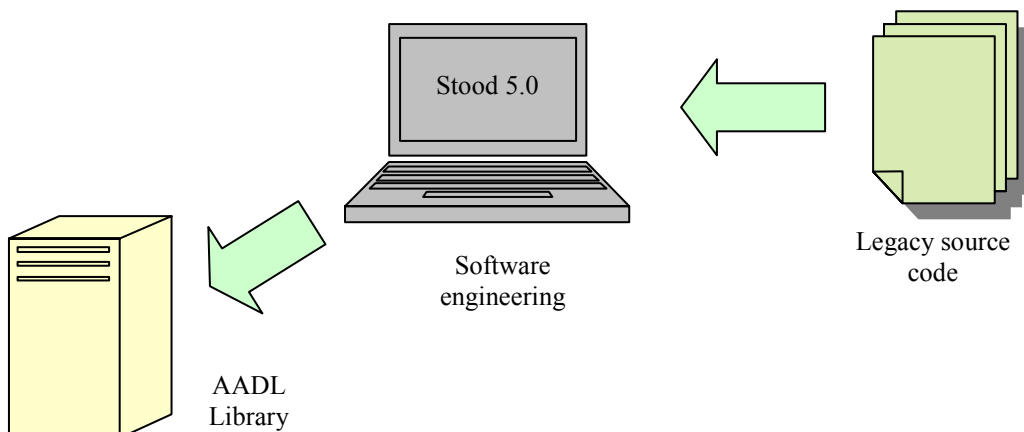
The normal continuation of this activity consists in performing the Software detailed design and coding steps that will lead to the automatic generation of ready to compile target source code and its synchronized design documentation.

However, this process may be easily enhanced by providing some feed back to the reusable components library. If the AADL has been chosen to describe these reusable components, the Software design tool must implement an AADL output transformation, so that the AADL code can be generated in a similar way as the target language source code. This AADL generation must of course be available for the whole design hierarchy, for a sub-tree only or even for a single component.



### 1.3 Creating a repository of components from legacy source code

In this use case, the Software design tool works in a reverse mode. It is not used to create a new application, but to build a library of reusable components from legacy target source code. Instead of directly converting target source code into AADL components, the use of an intermediate design tool can bring a very powerful help to reorganize the architecture and solve the multiple issues that arise when reversing legacy code that was not initially produced from a well structured design framework. Currently, this feature is available for Ada and C source code.



## 2 Model transformations

The features that have been presented above are implemented in the tool by formal model transformations based on logical rules. This technology has already been successfully used to implement various other plug-ins of the Stood 5.0 tool, to support the following formal transformations in a highly maintainable way:

- From software design model to source code generation, including real time features.
- From legacy source code to software design model.
- methodology and quality insurance: design rules, coding rules, requirements traceability, metrics, ...

The process to elaborate these rules is firstly to identify the common meta-model subset between the two ends of the transformation, then to create a point to point mapping between the corresponding concepts, and finally to specify the logical transformation rules for each of them. Thus, two sets of transformation rules have been specified to implement the AADL export and import features in Stood 5.0.

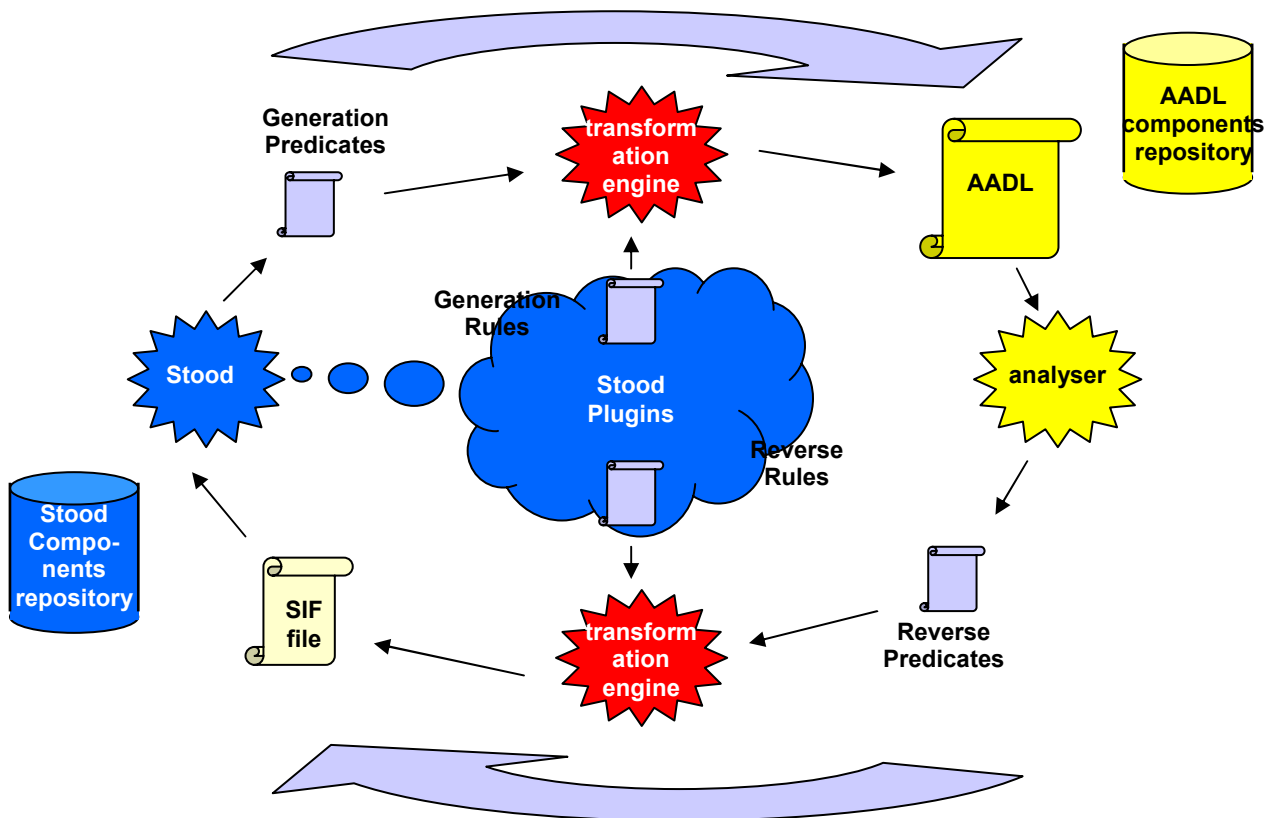
### 2.1 General principle

The two sets of rules (generation rules and reverse rules), expressed in prolog language, are located inside the plug-ins repository of the tool, and can be easily customized or extended independently if required.

Each time a “generation” transformation is activated by the user (AADL output), Stood automatically produces a set of prolog predicates that describe formally and exhaustively the current state of the Software design. A few additional predicates may also carry directives to the code generator. These predicates represent the “facts base” that is processed with the “generation rules base” by the prolog transformation engine. The result of this processing is a set of AADL source files.

In a similar way, each time a “reverse” transformation is activated by the user (AADL input), the dedicated AADL syntactic analyser that is provided with Stood 5.0, produces a set of prolog predicates that describe formally the current state of the AADL sources. This first step is a pure syntactic transformation, without any semantic interpretation. This “fact base” is then also processed by the “reverse rules base” by the same prolog transformation engine to perform the semantic transformation. The result of this processing is a SIF (Standard Interchange Format) file that is automatically imported into Stood 5.0 to create the corresponding design.

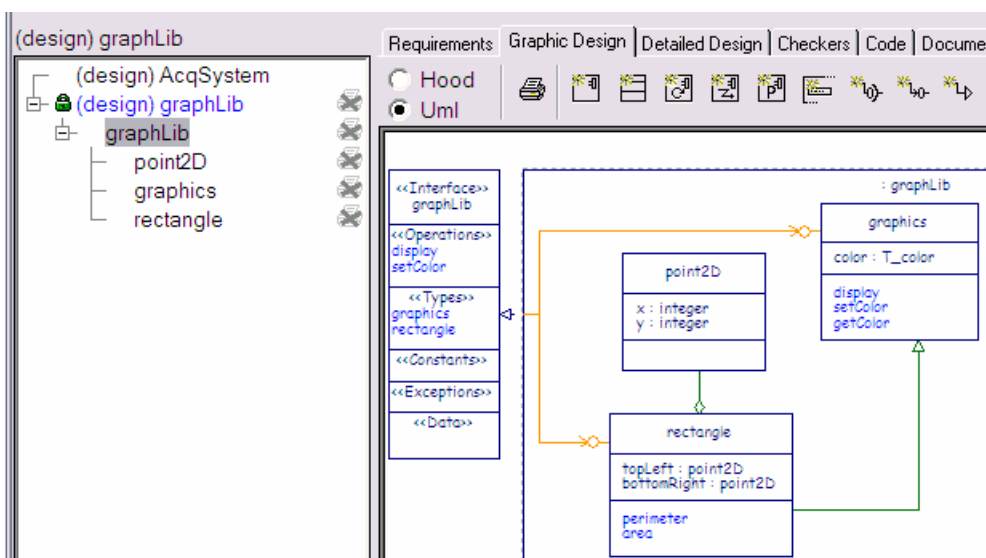
Although the AADL generation rules and AADL reverse rules are not fully symmetrical, we will simply show below a few examples of the result of the forward transformation only. A more complete description can be found in the Stood AADL Import/Export User Manual.



## 2.2 Transforming a UML Class diagram into an AADL package

This chapter shows an example of a UML 2.0 class diagram that has been graphically edited with Stood 5.0 and the corresponding AADL code that has been automatically generated from it. No additional information was required.

This first example includes three classes with attributes and operations, as well as an inheritance and a composition link. The container box is represented by an UML 2.0 component instead of usual packages, in order to highlight its interface and express the delegate relationships to the inner entities.



Corresponding AADL code consists in three child packages, one per class. Each package contains a data component type, a data component implementation and subprogram component types for operations. An option of the generator could be used to insert all these AADL components inside a single package.

```
PACKAGE graphLib::point2D
PUBLIC

  DATA point2D
  END point2D;

  DATA IMPLEMENTATION point2D.others
  SUBCOMPONENTS
    x : DATA integer;
    y : DATA integer;
  END point2D.others;

END graphLib::point2D;

PACKAGE graphLib::graphics
PUBLIC

  DATA graphics
  FEATURES
    display : SUBPROGRAM display;
    setColor : SUBPROGRAM setColor;
    getColor : SUBPROGRAM getColor;
  END graphics;

  DATA IMPLEMENTATION graphics.others
  SUBCOMPONENTS
    color : DATA T_color;
  END graphics.others;

  SUBPROGRAM display
  FEATURES
    me : IN OUT PARAMETER graphLib::graphics;
  END display;

  SUBPROGRAM setColor
  FEATURES
    me : IN OUT PARAMETER graphLib::graphics;
  END setColor;

  SUBPROGRAM getColor
  FEATURES
    me : IN OUT PARAMETER graphLib::graphics;
  END getColor;

END graphLib::graphics;

PACKAGE graphLib::rectangle
PUBLIC

  DATA rectangle EXTENDS graphLib::graphics
  FEATURES
    perimeter : SUBPROGRAM perimeter;
    area : SUBPROGRAM area;
  END rectangle;
```

```

DATA IMPLEMENTATION rectangle.others
SUBCOMPONENTS
  topLeft : DATA point2D::point2D;
  bottomRight : DATA point2D::point2D;
END rectangle.others;

SUBPROGRAM perimeter
FEATURES
  me : IN OUT PARAMETER graphLib::rectangle;
END perimeter;

SUBPROGRAM area
FEATURES
  me : IN OUT PARAMETER graphLib::rectangle;
END area;

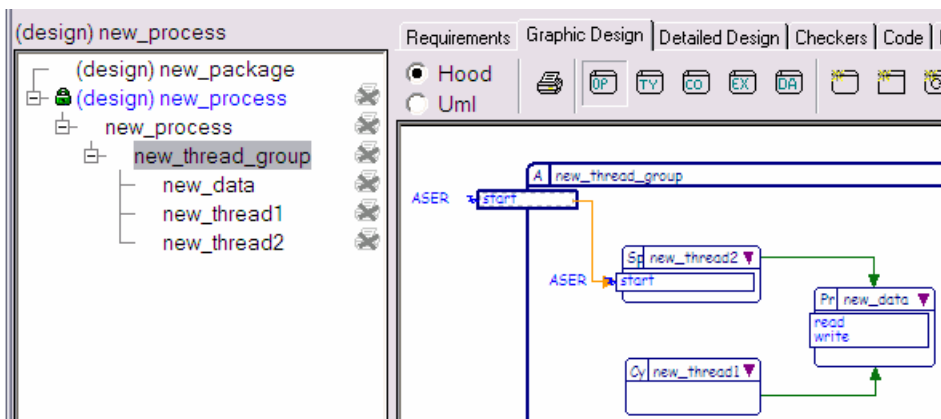
END graphLib::rectangle;

```

## 2.3 Transforming HRT objects into AADL components

As another example, this chapter shows a typical HRT diagram (that could be also displayed in UML with stereotyped components), and part of the corresponding AADL code that can be automatically generated by the tool.

This second example shows an active design (AADL process) containing a non terminal active object (AADL thread group) that includes an HRT sporadic object (AADL sporadic thread), an HRT cyclic object (AADL periodic thread) and an HRT protected object (AADL data with data subprograms).



The corresponding AADL code that is generated contains the various component types and implementations, as well as the subcomponents and the connections. Note that an event port has been created from a parameterless asynchronous operation (ASER), and that Use links to a shared protected object is translated into data access connections.

```

PROCESS new_process
FEATURES
  start : IN EVENT PORT;
END new_process;

PROCESS IMPLEMENTATION new_process.others
SUBCOMPONENTS
  new_thread_group : THREAD GROUP new_thread_group;
CONNECTIONS
  EVENT PORT start -> new_thread_group.start;
END new_process.others;

THREAD GROUP new_thread_group
FEATURES
  start : IN EVENT PORT;
END new_thread_group;

THREAD GROUP IMPLEMENTATION new_thread_group.others
SUBCOMPONENTS
  new_data : DATA new_data;
  new_thread1 : THREAD new_thread1;
  new_thread2 : THREAD new_thread2;
CONNECTIONS
  EVENT PORT start -> new_thread2.start;
  DATA ACCESS new_data -> new_thread1.new_data;
  DATA ACCESS new_data -> new_thread2.new_data;
END new_thread_group.others;

THREAD new_thread2
FEATURES
  start : IN EVENT PORT;
  new_data : REQUIRES DATA ACCESS new_data;
PROPERTIES
  Dispatch_Protocol => sporadic;
  Compute_Entrypoint => thread;
END new_thread2;

THREAD new_thread1
FEATURES
  new_data : REQUIRES DATA ACCESS new_data;
PROPERTIES
  Dispatch_Protocol => periodic;
  Compute_Entrypoint => thread;
END new_thread1;

DATA new_data
FEATURES
  read : SUBPROGRAM read;
  write : SUBPROGRAM write;
END new_data;

SUBPROGRAM read
FEATURES
  me : OUT PARAMETER new_data;
END read;

SUBPROGRAM write
FEATURES
  me : IN PARAMETER new_data;
END write;

```

### 3 Conclusion

These AADL transformation plug-ins are available in the current distribution of Stood 5.0 and can already be used by industrial projects or academic institutions. However, various enhancements are forecast to enlarge the support of the language by the tool, in connection with the users, the AADL committee and the related R&D projects. The main foreseen extensions are:

- an AADL legacy rules verification tool.
- an AADL graphical notation front end (currently, UML 2.0 and HRT-HOOD graphical notations are supported)
- Ada and C code generation rules as specified by the AADL annex (currently, Ada and C is automatically generated in compliance with the HOOD and HRT-HOOD rules).
- support of the COTRE language behavioral extension to the AADL
- support of distributed architectures in connection with the DDHRT cluster of ASSERT
- connect advanced real time architecture analysis and verification tools

### 4 References

1. HOOD User Group, *HOOD Reference Manual release 3.1*, Masson & Prentice-Hall, 1993.
2. HOOD User Group, *HOOD Reference Manual release 4.0*, HUG, 1995.
3. A. Burns, A. Wellings, *HRT-HOOD: A Structured Design Method for Hard Real-Time Ada Systems*, Elsevier, 1995
4. J.P. Rosen, *An Industrial Approach for Software Design*, HUG, 1997.
5. P. Dissaux, *HOOD4 and Ada95*, Proceedings DASIA, 1999.
6. T. Vardanega, *Development of On-Board Embedded Real-Time Systems: An Engineering Approach*, ESA Technical Report STR-260, 1999.
7. P. Dissaux, *Real-Time C Code Generation from a HOOD Design*, Proceedings DASIA, 2000.
8. P. Dissaux, *HOOD Patterns*, Proceedings DASIA, 2001.
9. P. Farail, P. Dissaux, *COTRE, A new Approach for Modelling Real-Time Software for Avionics*, Proceedings DASIA, 2002.
10. OMG. *UML 2.0 Final Adopted Specification*, ptc/03-08-02, 2002
11. P. Dissaux, *HOOD and AADL*, Proceedings DASIA conference in Prague, 2003.
12. P. Farail & P. Gauffillet, *The COTRE Project: How to model and verify Real Time Architecture?*, Proceedings ERTS conference in Toulouse, 2004.
13. P. Dissaux, *Using the AADL for mission critical software development*, Proceedings ERTS conference in Toulouse, 2004.
14. SAE, *Architecture Analysis and Design Language (AADL) AS5506, Version 1.0*, 2004
15. TNI Europe, *Stood/AADL Import/Export User Manual*, 2005.

The Stood tool and related documentation may be downloaded from:  
<http://www.tni-world.com>