

HOOD and AADL

P. Dissaux*

**TNI-Valiosys, Technopôle Brest-Iroise, BP 70801, F-29608 Brest Cedex, France
pierre.dissaux@tni-valiosys.com*

Abstract

This paper introduces a mapping between the Hierarchical Object Oriented Design (HOOD) method, widely used on many major European aerospace projects, and the Avionics Architecture Description Language (AADL), a candidate for a new American standard. The aim of the work presented here is to demonstrate that AADL, combined with parts of the new UML 2.0 notation set, can become an efficient and modern solution minimizing the migration effort for HOOD projects.

1 Introduction

1.1 The HOOD method

The HOOD method first appeared in 1987 to meet the requirements of the European Space Agency (ESA). The early versions already included a set of notations and precise design rules to support advanced software engineering concepts and Ada code generation rules. In 1992, the version 3.1 of the HOOD Reference Manual (HRM) was published. In 1995, two concurrent major releases of the method were issued : HOOD4, conducted by the French Space Agency (CNES) improved widely the support of true Object Orientation to enable Ada95 and C++ code generation. At the same time, a new ESA project produced a Hard Real Time version of the method, called HRT-HOOD, dedicated to embedded real-time systems, and providing a direct support for schedulability analysis. After several years of operational use of these different variants of HOOD by major European projects, it appeared that HOOD 3.1, HOOD4 and HRT-HOOD were fully compatible and reflected the implementation of the same basic principles for various application profiles.

Currently, the use of the HOOD method fully complies with the main software engineering standard recommended for mission critical development of avionics, space, defense and some ground transportation projects:

- DO-178B: Software Design Process
- ISO/IEC-12207: Software Architectural Design and Software Detailed Design
- ECSS-E40: Software top-level Architectural Design and Design of Software Items

The interest shown for AADL by several industrial supporters of HOOD (especially Airbus and ESA) makes it meaningful to study in details the similarities and the differences between these two approaches.

1.2 The Avionics Architectural Description Language

The Avionics Architecture Description Language (AADL) standard was prepared by the Society of Automotive Engineers (SAE) AS-2C Architecture Description Language Subcommittee, Embedded Computing Systems Committee, Aerospace Avionics Systems Division. The AADL standard is based on MetaH, an avionics architecture description language and toolset developed at Honeywell Laboratories under the sponsorship of the US Defense Advanced Research Projects Agency (DARPA) and US Army Aviation and Missile Command (AMCOM).

The Avionics Architecture Description Language (AADL) is a computer language used to describe the software and hardware components of an avionics system and the interfaces between those components. The language is used to describe the structure of an avionics system as an assembly of software and hardware components. The language can describe functional interfaces to components (such as data inputs and outputs) and non-functional aspects of components (such as timing). The language can describe how components are combined (such as how data inputs and outputs are connected or how software components are allocated to hardware components). The AADL was developed to meet the special needs of avionics systems. In particular, the language can describe standard control and data flow mechanisms used in avionics systems, and the language can describe important non-functional aspects such as timing requirements, fault and error behaviors, time and space partitioning, and safety and certification properties.

1.3 Comparison between HOOD and AADL

At first look, many similarities in terms of general motivation and technical choices appear when comparing the AADL standard draft (v0.6) and the HOOD method (merge of v3.1, v4.0 and HRT):

- They are both hierarchical and compositional
- They both propose a precise definition of the interfaces (provided and required)
- They both handle low level software items (data types, data, subprograms, ...)
- They both support real-time paradigms (threads, communication, ...)
- They both rely on a textual language (although HOOD also proposes a graphical notation)
- They both mainly target Ada and C

However, a point to point analyse raises questions about possible semantical discrepancies that would have to be solved. These differences have been highlighted in next chapter, in order to help finding appropriate solutions. Nevertheless, for the large majority of the points that have been studied, a quite direct mapping between the HOOD and the AADL concepts has been found. This mapping is presented in a set of tables where we propose a likely HOOD equivalent for most AADL concept. These mapping rules have then been used to implement an AADL code generator and an AADL parser for the Stood tool.

This mapping will be probably updated for the next versions of the AADL draft of standard, and of course will be finalized for the first release of the standard when available.

2 Mapping of concepts

2.1 AADL Components

A HOOD System_Configuration represents a software project. It refers to one or several cooperative programs (Design), libraries and software interface with the hardware (Environment), and generic components (Generic). Each Design may become an Environment within the scope of another Design of the same System_Configuration. This is the "logical view" of the project. In addition, it may contain one or several Virtual_Nodes describing a distributed architecture of processors and an allocation table for the elements of the logical view. This is the "physical view" of the project which is only needed for distributed systems, and not very often used in practice.

A Design is the root of a hierarchy of Modules. Each Module is a well identified subset of the Design that is refined in an iterative way, following a top-down decomposition process. This process must comply with the "low-coupling" and "high consistency" rules promoted by the HOOD method.

A Module may be Passive or Active. If it is Active, it contains its own thread of control. Two specialized kinds of Active Modules have been defined to support Hard Real-Time projects: Cyclic Objects to represent periodic activity, and Sporadic Objects that are triggered by an explicit event. A specialized kind of Passive Module has been defined to encapsulate shared data and their appropriate access procedures, they are called Protected Objects. There are other kinds of Modules like Classes and Instances_Of generic.

The main difference that appears at this level is:

- (DIFF-001) AADL covers both hardware and software architectures, whereas HOOD is fully dedicated to pure software development, with access to hardware drivers only.

However, the following mapping for AADL components is proposed:

AADL Component	HOOD
system	system configuration
processor	virtual node (software allocated to a processor)
device, bus, memory	environment (software interface, driver)
process	design
package	passive module
thread	active module
thread {scheduling_protocol => periodic }	cyclic object (HRT)
thread {scheduling_protocol => sporadic }	sporadic object (HRT)
data	protected object (HRT)

In HOOD, each Module (including the Design), has an interface and a body. The interface consists of the Provided_Interface that lists the declaration of all the software elements that are implemented by the Module and made visible to be used by other Modules, and the Required_Interface that lists the references to all the remote software elements that are required to implement the Module. The body is called the Internals of the Module and contains either child Modules (for Non_Terminal Module) or a list of declaration of private software elements and all the implementation (for Terminal Module).

Additionally, HOOD only distinguishes between the Modules "type" and Modules "instances" in two particular cases only. The general case being that a Module is handled as an Object, that is, the unique instance of an anonymous "type". The first particular case is when the Module "type" is described by a Class (like in UML), but then, instances of this Class (which are in effect only instances of the main data Type provided by the Class) becomes Data, Constants, etc... embedded somewhere inside another Module. The other particular case is when the Module "type" is described by a Generic, then, instances of this Generic (which are in effect only instances of the formal parameters) becomes other Modules called Instance_Of. HOOD also supports Generic Classes (similar to C++ templates) that need to be instantiated twice. The only extension mechanism offered by HOOD is the Class Inheritance.

So, the main differences with AADL are:

- (DIFF-002) In AADL, a Component Type may have several Component Implementations, whereas in HOOD, the mapping is one to one.
- (DIFF-003) In AADL, a Component Implementation may have both Features and Subcomponents, whereas in HOOD, this is exclusive and Non_Terminal Modules are "empty shells".
- (DIFF-004) In AADL, Component Types and Component instances are distinguished, whereas in HOOD, this is only true for Classes and Generics.
- (DIFF-005) In AADL, any Component Type may Extend another one, whereas in HOOD, only Classes may be extended with an appropriate Inheritance relationship.

The resulting mapping at that level may be summarized as follow:

AADL Components details	HOOD
component type:	
features	provided interface of the module
required subcomponents	required interface of the module
component implementation:	
features	internals of a terminal module
subcomponents	children of a non terminal module
extends	inheritance (allowed only for classes)

2.2 AADL Features

A HOOD Module may contain references and declarations to, and/or implementation of, software elements. These elements are: Operations, Exceptions, Types and Constants in a Provided_Interface, and Operations, Types, Constants and Data in the Internals. Operations declaration may specify Parameters and Types declaration may specify Attributes. Constants, Data, Parameters and Attributes are instances of a data Type. Operations may raise and handle Exceptions. As Provided Data are not allowed in HOOD, data flows can only be propagated between two Modules along client/server functional calls.

By default, the execution request for an Operation is unconstrained. But it is possible to specify one or several Operation Constraints to describe more precisely the interaction between a client and a server. There are several kinds of Constraints:

- Protocol Constraints to specify the synchronization protocol between two concurrent threads of execution. These constraints are: ASER (asynchronous), LSER (synchronous, acknowledge), HSER (synchronous, wait-reply). The additional

constraint TO specifies a time-out for ASER and LSER, and hardware interrupts are identified by the additional constraint BY_IT attached to an ASER constraint.

- State Constraints to specify the receptivity of a service regarding the current State of the server.
- Concurrency Constraints to manage mutual exclusion.

The main differences with AADL are:

- (DIFF-006) In AADL, remote data can be directly accessed asynchronously, whereas in HOOD, only provided Constants are allowed, and R/W dataflows are supported by subprogram calls only.
- (DIFF-007) In AADL, data other than ports are not declared, whereas in HOOD, all the software elements must be declared either in the Provided_Interface, or in the Internals.

AADL Features	HOOD
data type	type
port:	
data	provided data (allowed only for constants)
event	parameters of a provided operation ASER constraint of a provided operation exception
event data	ASER constraint and parameters of a prov. operation
subprogram	non constrained operation HSER or LSER constrained provided operation

2.3 AADL Connections

At architectural level, the interactions between HOOD Modules are described by Use relationships on both functional and structural views. A functional Use relationship defines a client/server interaction between Operations of the two Modules. They are the support for control flows, data flows (related to the parameters of the called Operations) and exception flows (related to the Exceptions that may be raised by the called Operations). A structural relationship defines a Type dependency (instanciation, aggregation and inheritance for Classes) between the two Modules. They are the support for the definition of Attributes (for any structured Type) and super-Classes (for Classes).

In HOOD, non Terminal Modules are empty shells. That means that any element (Type, Constant, Operation or Exception) declared in the Provided_Interface of a non Terminal Module must be Implemented_By a element of the same kind in the Provided_Interface of one of the child Modules.

The differences with AADL are:

- (DIFF-009) In AADL, data connections may be specified, whereas in HOOD, only Operation calls are explicitly defined.
- (DIFF-010) In AADL, use of a type is specified by a dot notation in identifiers, whereas in HOOD, Type_Use and Implemented_By links must also be defined for Types.

AADL Connections	HOOD
	use relationship:
data	N/A
event	operation call without parameters
event	raised exception
event data	operation call with parameters
subprogram	operation call
component ports to subcomponent ports	implemented by relationship
dot notation	type use relationship

2.4 AADL Properties

HOOD offers a standard documentation structure to describe the details of each Module. This structure is called the ODS (Object Description Skeleton) and provides lower level details about the Module and its elements. Typical information supported by the ODS is:

- Textual sections to justify the design choices and the requirements traceability
- Real-Time attributes (period, priority, deadline, worst case execution time, etc...)
- The list of required remote elements (Required_Interface)
- Textual comment for each Operation, Type, etc...
- Source code for the declaration of each Type, Constant, or Data element
- Source code for the OPCS (procedural code of an Operation) and the OBCS (behavioral code of a Module)
- etc...

The standard contents of the HOOD ODS must be compared to the list of properties defined by AADL for each kind of component or feature. Following tables just show a first mapping. A deeper analyse would be necessary to define a more complete and correct mapping, and a list of differences:

AADL	HOOD
software components:	
Source Text	defined by code generation rules
Source Language	pragma Target Language
Source Code Size	N/A
Source Data Size	N/A
Source Mapping	defined by code generation rules
package:	
Source Name	defined by code generation rules
thread:	
Source Stack Size	N/A
Source Name	defined by code generation rules
Scheduling Protocol	deduced from the module kind (cyclic, sporadic, ...)
Period	HRT Real Time Attribute: Period
Initialize Compute Time	HRT Real Time Attributes: Budget & WCET
Compute Time	
Recover Compute Time	
Initialize Deadline	HRT Real Time Attribute: Deadline
Deadline	
Recover Deadline	
process:	
Load Deadline	N/A

AADL	HOOD
features	
Source Mapping	defined by code generation rules
Source Name	defined by code generation rules
data type	
Source Size	N/A
port	
Event Handling	managed by operation constraints (ATC, ...)
mutex	
Protocol	concurrency operation constraints (PSER, PAER, ...)
subprogram	
Compute Time	HRT Real Time Attributes: Budget & WCET
Stack Size	N/A

2.5 AADL Behaviors

The Internals of a Terminal Module contain the procedural code associated to each Provided or Internal Operation, within a structure called OPCS (Operation Control Structure). If there is at least one Operation that has a Protocol or State Constraint, the Internals of a Terminal Module will also contain the behavioral code associated to the corresponding thread and/or states-transitions model, within another structure called OBCS (Object Control Structure).

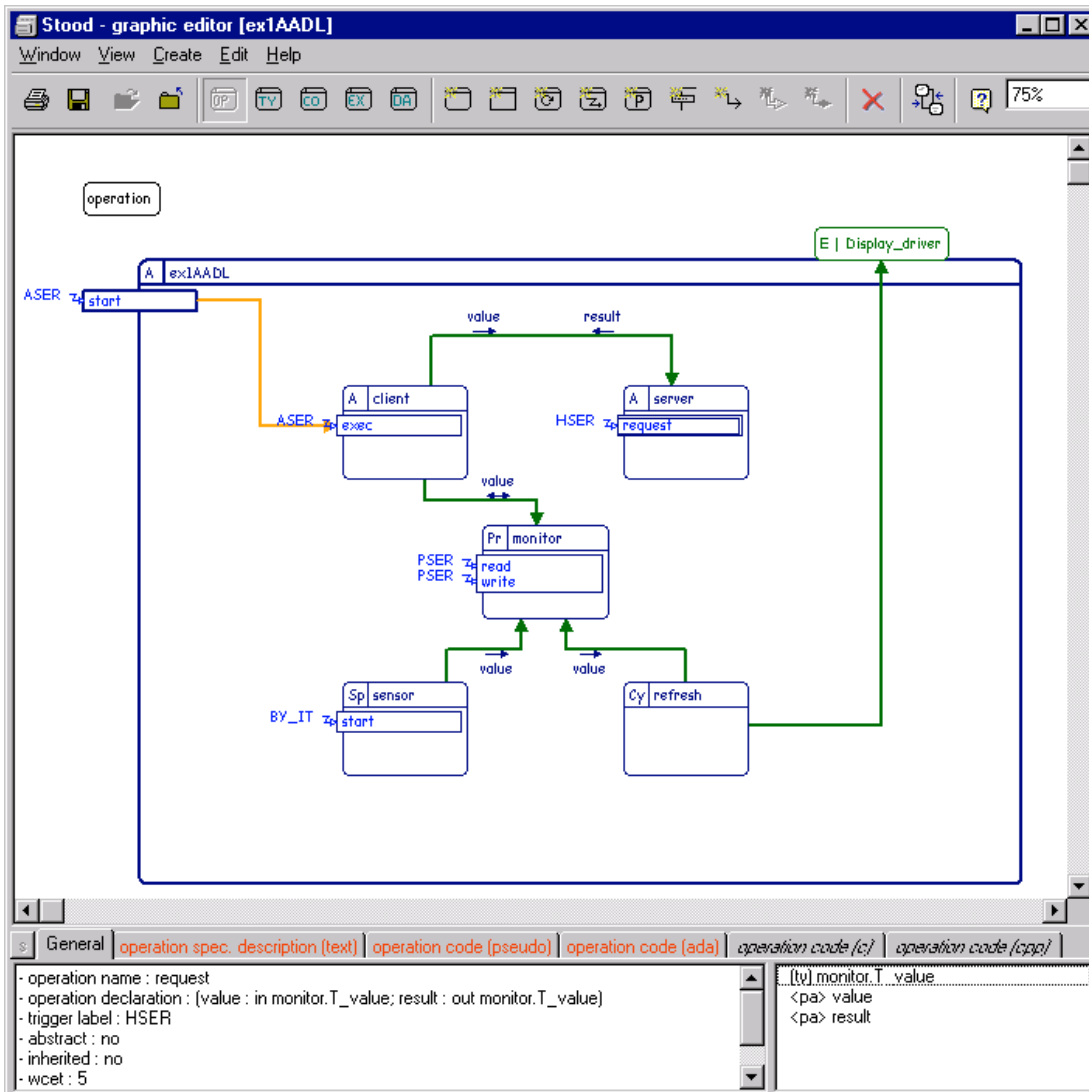
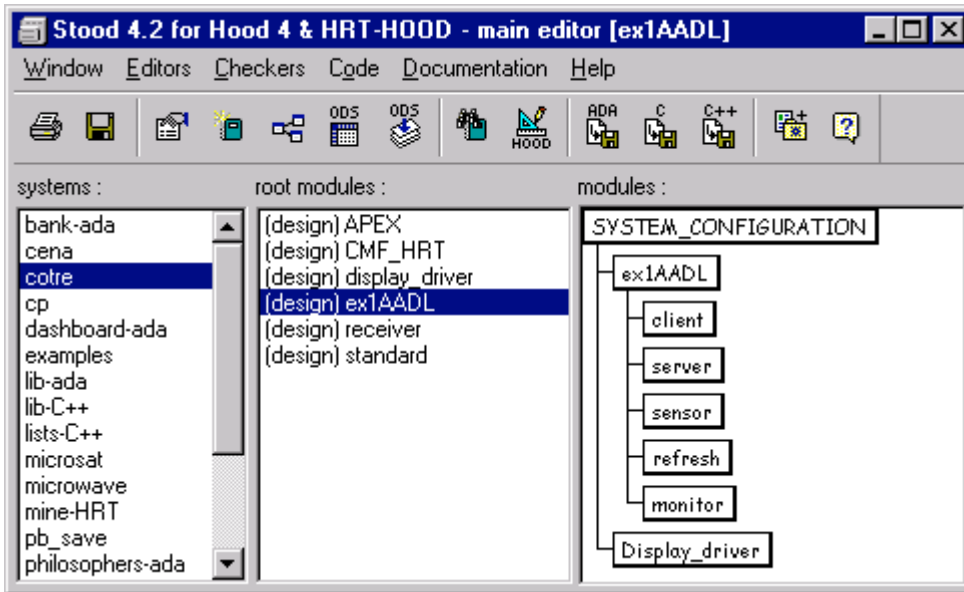
Internal Data may be shared by all the elements contained by the Internals of the Module. They can be used as State variables for the states-transitions model implemented in the OBCS of a Terminal Module. The only states that need to be specified in a HOOD states-transitions model are those defining areas of receptivity for the provided Operations. The execution request for a provided Operation becomes an event that triggers a transition (in the OBCS), before executing the appropriate code (in the OPCS). In HOOD, a states-transitions model can also be used to specify the execution modes for a whole Design. In that case, all the real-time attributes must have a known value for each mode.

A deeper analyse would be necessary to compare AADL and HOOD states-transitions models. Here is a first draft only:

AADL	HOOD
	states-transitions defined in a terminal modul:
state	state (subset of valid values of internal data)
transition	STATE constrained provided operation
	states-transitions defined in the root module:
state	mode (for the whole design)
transition	N/A

3 A simple example

3.1 The HOOD graphical notation



3.2 The HOOD textual notation

Here is a incomplete HOOD textual description for the design shown above. When complete, this description contains all the design information (architecture, code, documentation, real-time attributes). The used syntax is the Standard Interchange Format (SIF), defined by the HOOD Reference Manual, and which input/output is provided by the Stood tool.

```
OBJECT SYSTEM_CONFIGURATION IS PASSIVE
INTERNALS
  OBJECTS
    ex1AADL;
    Display_driver;
    OS;
END SYSTEM_CONFIGURATION
```

```
OBJECT ex1AADL IS ACTIVE
PRAGMA main(operation_name => start)
PROVIDED_INTERFACE
  OPERATIONS
    start
OBJECT_CONTROL_STRUCTURE
  CONSTRAINED_OPERATIONS
    start CONSTRAINED_BY ASER ;
REQUIRED_INTERFACE
  OBJECT Display_driver;
  OPERATIONS
    refresh;
INTERNALS
  OBJECTS
    client;
    server;
    sensor;
    refresh;
    monitor;
  OPERATIONS
    start IMPLEMENTED_BY client.exec;
END ex1AADL
```

```
OBJECT client IS ACTIVE
PROVIDED_INTERFACE
  OPERATIONS
    exec
OBJECT_CONTROL_STRUCTURE
  CONSTRAINED_OPERATIONS
    exec CONSTRAINED_BY ASER ;
REQUIRED_INTERFACE
  OBJECT server;
  OPERATIONS
    request;
  OBJECT monitor;
  OPERATIONS
    read;
    write;
DATAFLOWS
  result <= server;
  value => server;
  value <=> monitor;
INTERNALS
  OBJECT_CONTROL_STRUCTURE
    CODE --| Ada code for the task |--
  OPERATION_CONTROL_STRUCTURES
    OPERATION start
      CODE --| Ada code for the operation |--
    END_OPERATION start
END client
```

```

OBJECT server IS ACTIVE
  PROVIDED_INTERFACE
    OPERATIONS
      request(value : in monitor.T_value ; result : out monitor.T_value )
  OBJECT_CONTROL_STRUCTURE
    CONSTRAINED_OPERATIONS
      request(value : in monitor.T_value ; result : out monitor.T_value ) CONSTRAINED_BY HSER ;
  REQUIRED_INTERFACE
    OBJECT monitor;
    TYPES
      T_value;
END server

```

```

OBJECT sensor IS SPORADIC
  REAL TIME ATTRIBUTES
    MIN_ARRIVAL_TIME 10
    DEADLINE 10
    PRIORITY 1
  PROVIDED_INTERFACE
    OPERATIONS
      start
  OBJECT_CONTROL_STRUCTURE
    CONSTRAINED_OPERATIONS
      start CONSTRAINED_BY ASER BY_IT --|10|--;
  REQUIRED_INTERFACE
    OBJECT monitor;
    OPERATIONS
      write
  DATAFLOWS
    value => monitor;
  INTERNALS
    OPERATIONS
      thread
END sensor

```

```

OBJECT refresh IS CYCLIC
  REAL TIME ATTRIBUTES
    PERIOD 10
    DEADLINE 5
    PRIORITY 5
  REQUIRED_INTERFACE
    OBJECT monitor;
    OPERATIONS
      read
  OBJECT Display_driver;
  OPERATIONS
    refresh
  DATAFLOWS
    value <= monitor;
    value => Display_driver;
  INTERNALS
    OPERATIONS
      thread
END refresh

```

```

OBJECT monitor IS PROTECTED
  REAL TIME ATTRIBUTES
    CEILING_PRIORITY 1
  PROVIDED_INTERFACE
    TYPES
      T_value --| type T_value is range 0..10; |--
    OPERATIONS
      read(value : out T_value )
      write(value : in T_value )
  INTERNALS
    DATA
      Buffer --| buffer : T_value := 0; |--
    OPERATION_CONTROL_STRUCTURES
      OPERATION read(value : out T_value )
        CODE --| begin value := buffer; |--
      END_OPERATION read(value : out T_value )
      OPERATION write(value : in T_value )
        CODE --| begin buffer := value; |--
      END_OPERATION write(value : in T_value )
END monitor

```

3.3 The AADL specification

This is an attempt to translate the previous HOOD design into an AADL specification. The goal is to specify the proper rules to automatically generate AADL code from a HOOD tool, and to be able to parse AADL code to build a correct HOOD design.

All the identifiers written in *italic* style represent entities that are not explicitly named in the HOOD model, but that are needed by the AADL syntax. The identifier name *default* is often used for that purpose.

```
SYSTEM system_configuration IS
  Start : IN EVENT;
END system_configuration;

SYSTEM IMPLEMENTATION system_configuration.default IS
  ex1AADL : PROCESS ex1AADL.default ;
  Display_Driver : PROCESS display_driver.default;
  OS : SYSTEM system.default ;
CONNECTIONS
  EVENT DATA Display_Driver.data <- ex1AADL.data;
  EVENT ex1AADL.start <- start;
  EVENT ex1AADL.interrupt <- system.sensor_interrupt;
END system_configuration.default ;
```

```
PROCESS ex1AADL IS
  start : IN EVENT;
  data : OUT EVENT DATA monitor.T_value;
  interrupt : IN EVENT;
END ex1AADL;
```

```
PROCESS IMPLEMENTATION ex1AADL.default IS
  client : THREAD client(the_monitor => monitor ).default;
  server : THREAD server.default;
  sensor : THREAD sensor(the_monitor => monitor ).default {access => write};
  refresh : THREAD refresh(the_monitor => monitor ).default {access => read};
  monitor : PACKAGE monitor.default;
CONNECTIONS
  EVENT client.exec <- start;
  SUBPROGRAM server.request <- client.request;
  EVENT sensor.start <- interrupt;
  EVENT DATA data <- refresh.value;
END ex1AADL.default;
```

```
THREAD client IS
  exec : IN EVENT;
  request : CLIENT SUBPROGRAM (IN DATA value : monitor.T_value, OUT DATA result : monitor.T_value);
REQUIRES
  the_monitor : MONITOR monitor;
END THREAD;
```

```
THREAD IMPLEMENTATION client.default IS
PROPERTIES
  Scheduling_protocol => aperiodic ;
END client.default;
```

```
THREAD server IS
  request : SERVER SUBPROGRAM (IN DATA value : monitor.T_value, OUT DATA result : monitor.T_value);
END server;
```

```
THREAD IMPLEMENTATION server.default IS
PROPERTIES
  Scheduling_protocol => server;
END server.default;
```

```
THREAD sensor IS
  start : IN EVENT;
REQUIRES
  the_monitor : MONITOR monitor;
END sensor;
```

```
THREAD IMPLEMENTATION sensor.default IS
PROPERTIES
  Scheduling_protocol => sporadic ;
END sensor.default;
```

```
THREAD refresh IS
  value : OUT DATA monitor.T_value;
REQUIRES
  the_monitor : MONITOR monitor;
END refresh;
```

```
THREAD IMPLEMENTATION refresh.default IS
PROPERTIES
  Scheduling_protocol => periodic ;
  Period => 100 ms ;
END refresh.default;
```

```
DATA monitor IS
  T_value : TYPE;
  read : SUBPROGRAM (value : out T_value);
  write : SUBPROGRAM (value : in T_value);
END monitor;
```

```
DATA IMPLEMENTATION monitor.default IS
END monitor.default;
```

4 References

1. RTCA, *Software Considerations in Airborne Systems and Equipment Certification (DO-178B)*, 1992.
2. ISO/IEC, *Information technology, Software life cycle process (ISO/IEC 12207)*, 1995
3. ECSS, *Space Engineering: Software (ECSS-E40B)*, ESA Publication, 2000.
4. HOOD User Group, *HOOD Reference Manual release 3.1*, Masson & Prentice-Hall, 1993.
5. HOOD User Group, *HOOD Reference Manual release 4.0*, HUG, 1995.
6. A. Burns, A. Wellings, *HRT-HOOD: A Structured Design Method for Hard Real-Time Ada Systems*, Elsevier, 1995
7. J.P. Rosen, *An Industrial Approach for Software Design*, HUG, 1997.
8. P. Dissaux, *HOOD4 and Ada95*, Proceedings DASIA, 1999.
9. T. Vardanega, *Development of On-Board Embedded Real-Time Systems: An Engineering Approach*, ESA Technical Report STR-260, 1999.
10. P. Dissaux, *Real-Time C Code Generation from a HOOD Design*, Proceedings DASIA, 2000.
11. P. Dissaux, *HOOD Patterns*, Proceedings DASIA, 2001.
12. P. Farail, P. Dissaux, *COTRE, A new Approach for Modelling Real-Time Software for Avionics*, Proceedings DASIA, 2002.
13. SAE, *Draft Avionics Architecture Description Language (AADL) AS5506, AS-2C*, 2003

The Stood tool and documentation about the HOOD method may be downloaded from:

<http://www.tni.fr/contact/formgb.htm?prod=stood>

A simplified AADL generator and parser are provided with the tool.