

Stood 5.3

AADL

User Manual

Pierre Dissaux
Ellidiss Technologies

Contents

1	Introduction.....	5
2	General information.....	6
2.1	Installation of Stood.....	6
2.2	Start and quit a session.....	7
2.3	Overview of Stood.....	8
2.4	Stood life cycle selector.....	9
2.5	AADL graphical editor.....	9
2.5.1	AADL tool bar.....	10
2.5.2	AADL package or component contextual menu.....	10
2.5.3	AADL sub package or subcomponent contextual menu.....	11
2.5.4	AADL feature contextual menu.....	11
2.6	AADL textual editor.....	12
3	Edition of AADL models.....	15
3.1	Load existing models.....	15
3.1.1	Open a project.....	15
3.1.2	Load a design model.....	15
3.2	Create new models.....	16
3.2.1	Create a new project.....	16
3.2.2	Add or remove design models in a project.....	17
3.2.3	Create a new design model.....	17
3.2.4	Create a new design model from an existing AADL specification.....	17
3.2.5	Create a new design from existing Ada or C source files.....	18
3.3	AADL packages.....	18
3.4	AADL components.....	20
3.4.1	AADL Processes.....	22
3.4.2	AADL Threads.....	24
3.4.3	AADL Thread groups.....	25
3.4.4	AADL Data.....	26
3.4.5	AADL Subprogram components.....	27
3.5	AADL features.....	28
3.5.1	AADL Ports.....	28
3.5.2	AADL Feature groups.....	29
3.5.3	AADL Subprogram features.....	29
3.6	AADL connections.....	30
3.7	AADL properties.....	30
3.7.1	Stood property set.....	31
3.8	AADL modes.....	32
3.9	AADL flows.....	33
3.10	Behavior Annex.....	36
4	Processing of AADL models.....	38
4.1	Generate design verification reports.....	38

4.2Generate textual AADL code.....	39
4.3Generate Ada source code.....	41
4.4Generate C source code.....	42
4.5Generate design documentation.....	42

1 Introduction

The Architecture Analysis and Design Language (**AADL**) standard document was prepared by the **SAE AS-2C Architecture Description Language Subcommittee**, Embedded Computing Systems Committee, Aerospace Avionics Systems Division. Release 1.0 of the **AADL** standard (**SAE AS5506**) has been issued in November 2004, and Release 2.0 (**SAE AS5506A**) in January 2009. A set of annexes (**SAE AS5506/2**) has been published in January 2011 that includes in particular the Behavior Annex.

AADL is a language used to describe the software and hardware components of a system and the interfaces between those 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. More detailed information about this language may be found at: www.aadl.info.

Stood is a software design tool that is used for the same families of mission critical systems as those for which **AADL** has been developed. Like **AADL**, **Stood** promotes Model Driven Engineering (**MDE**) together with a Component Based modelling approach. This manual describes the features that have been added to **Stood**, in order to let software designers benefit from **AADL**. Some of the most important advantages of **Stood** are that it offers a very good support of the modelling process, and brings a large set of development features that have already been in use on many large scale industrial projects. With the **AADL** customization of **Stood** it is possible to:

- Import legacy specifications written in textual **AADL** (`.aadl` files).
- Edit graphically new or imported **AADL** models and generate corresponding textual **AADL** specification.
- Transform software **AADL** architecture (**AADL Process**) into **HOOD** designs to perform detailed software design activities.
- Produce design documentation, **Ada** and **C/C++** source code from design models.

This manual is not a complete User Manual for **Stood**. It only provides a brief description of the main features that can be useful for **AADL** projects developments.

2 General information

2.1 *Installation of Stood*

Stood can be easily and quickly installed on a **Windows** or a **Unix** workstation. On **Windows**, follow the instructions given by the guided installation program. On **Unix**, uncompress and expand the installation archive in an appropriate directory.

On **Windows**, default installation procedure will associate the following file extensions in the registry:

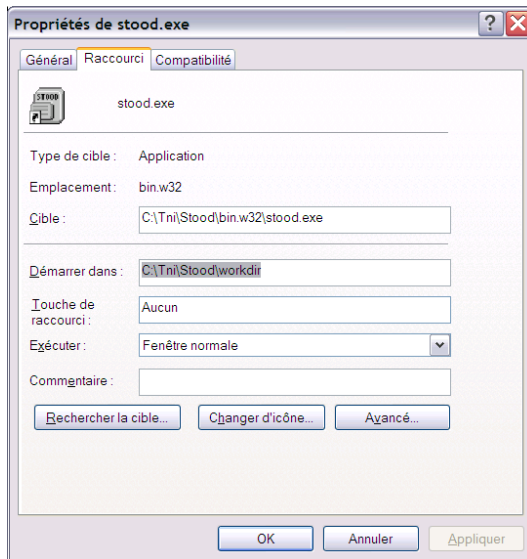
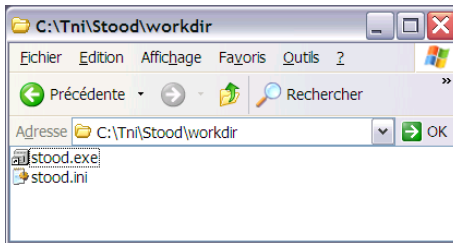
.*sync* **Stood** project file
.sto **Stood** design model file
.sts **Stood** command file

In addition to this installation, it is recommended, but not mandatory, to also create one or several working directories, properly separated from the installation directory. These working directories will be used to store user's models and set up user's specific initialization properties for **Stood**.

A **Stood** working directory should ideally contain:

- A link (**Windows** shortcut or **Unix** shell script) to the **Stood** executable file. **Windows** shortcuts must have their start up property set to the current working directory, so that new models will be created there.
- a *stood.ini* (on **Windows**) or *.stoodrc* (On **Unix**) initialization file, containing only the user's properties that differ from the default initialization file located in the installation directory.
- a set of project files and design model directories.

A typical empty **Stood** working directory on **Windows** looks like as shown in the following pictures.



2.2 *Start and quit a session*

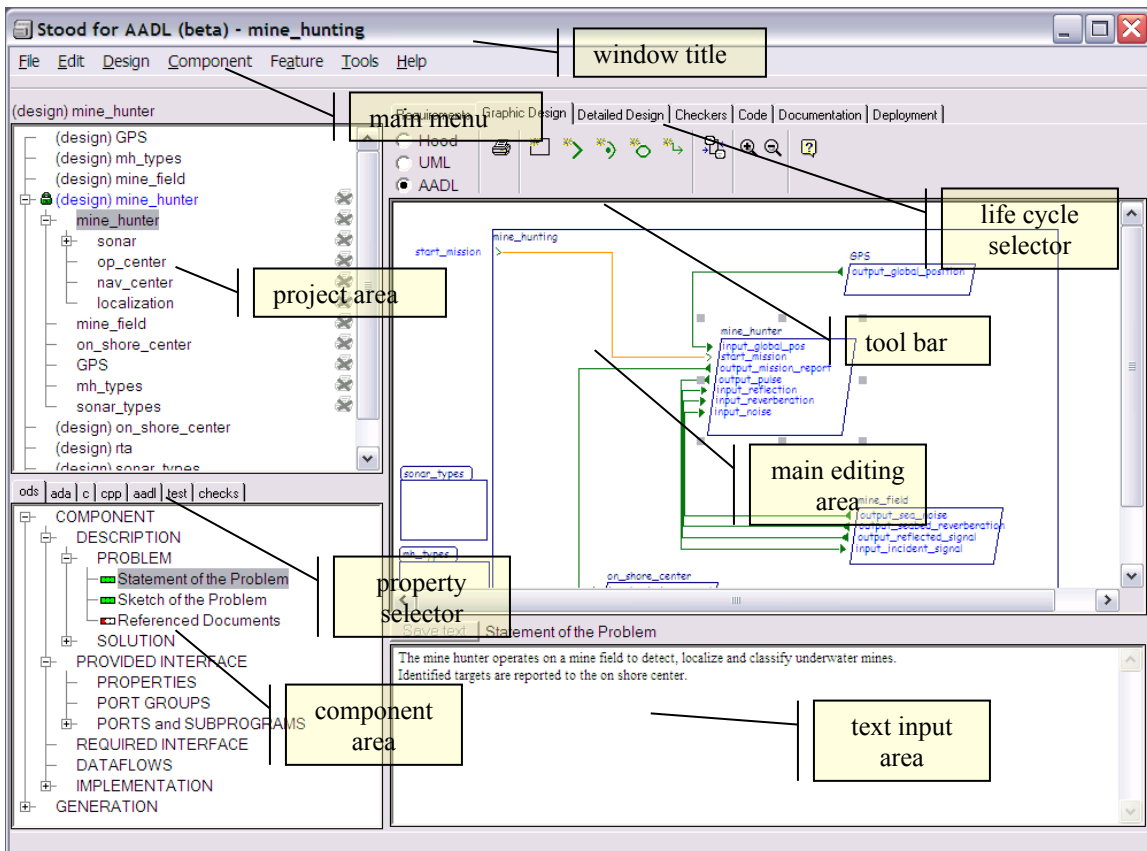
It is recommended to launch **Stood** from one of the pre-set working directories (refer to paragraph 2.1), by double-clicking on the **Stood** executable shortcut, or on a local project file (.SYC). It is of course also possible to launch the tool from the **Windows** start up menu, or even directly from the installation directory. However, in these last cases, if the default initialization file is not customized, new projects and design models will be created within the installation directory.

Stood is a multi user environment. It is thus possible to launch several concurrent sessions of the tool on a same project. **Stood** automatically manages the protection locks for the shared parts of the project. To close a session, simply use *File/Quit* in the main menu.

2.3 Overview of Stood

After the initialization process has completed, **Stood** main window is displayed. This paragraph provides information about the organization and the main features of this window. This window is composed of the following parts:

- a *window title* showing a customizable message and the name of the current project.
- a *main menu* bar grouping all the main non graphical possible user actions.
- a *project area* showing the structure of the current project.
- a *component area* showing details for the selected component in the Project area.
- a *property selector* acting as a filter for components details.

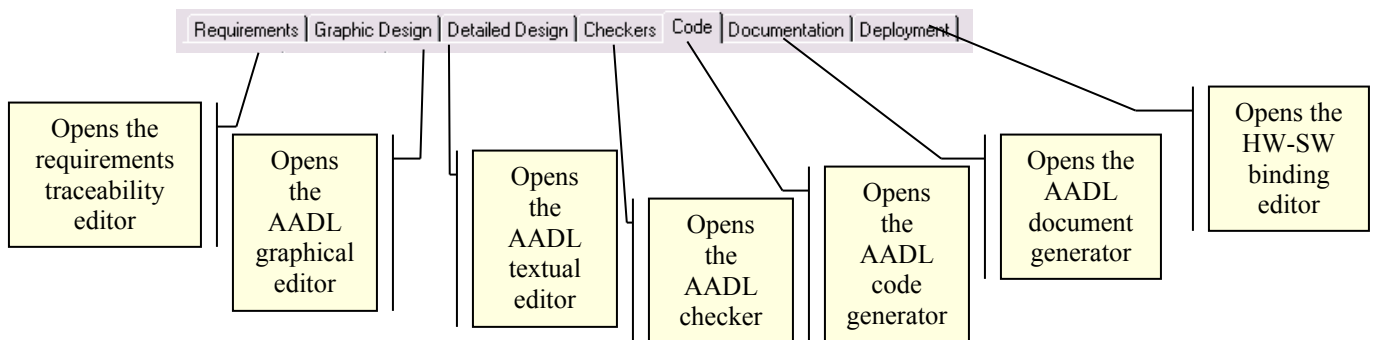


- a *main editing area* where the various graphical and textual editors are plugged.
- a *text input area*, that is displayed when a graphical editor is shown in the Main editing area.
- a contextual and customizable tool bar.
- a *project life cycle selector*, to be used as a switch for the main activities of the development process.

The *project area*, *component area*, *text input area* and *main editing area* also offer a *contextual menu*, that usually simply recalls items of the *main menu*.

2.4 *Stood life cycle selector*

Stood covers all the life cycle steps from requirements capture to target source code generation. **Stood** features that are available to support these various activities can be activated thanks to the life cycle selector.

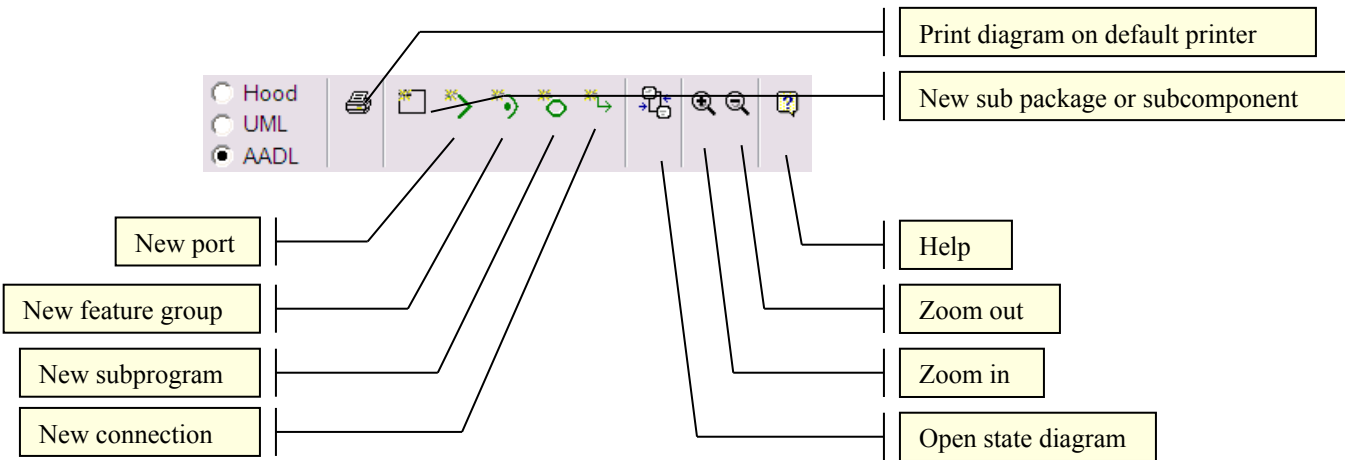


2.5 *AADL graphical editor*

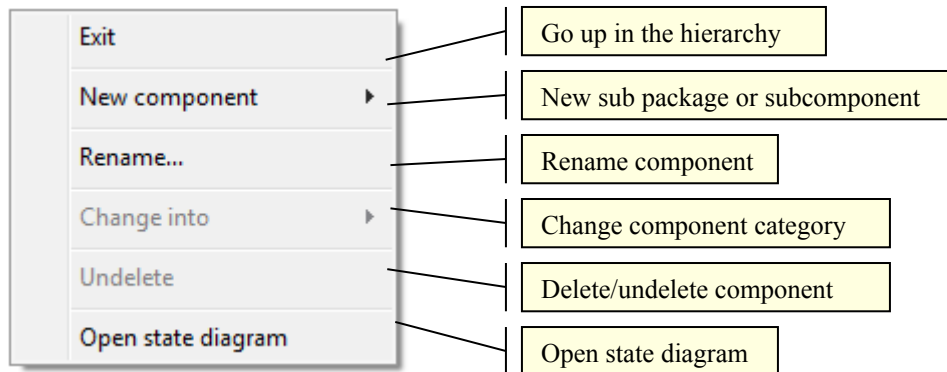
The **AADL** graphical editor of **Stood** is mainly composed of a button tool bar and a contextual menu in the graphical edition area. The contextual menu varies according to the current selection, depending on it is:

- an **AADL** component or package
- an **AADL** subcomponent or sub package
- an **AADL** feature

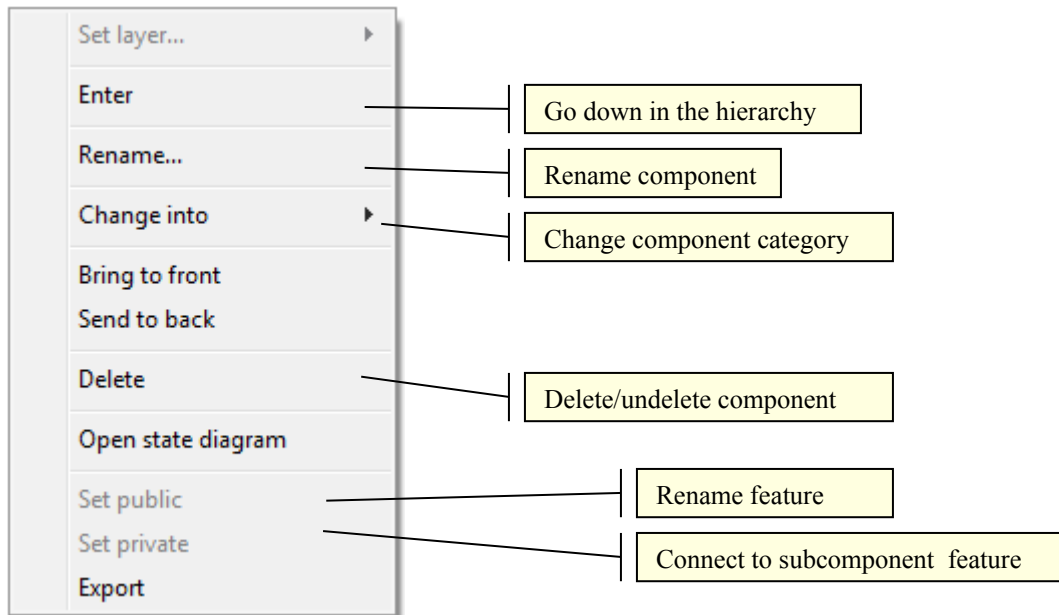
2.5.1. AADL tool bar



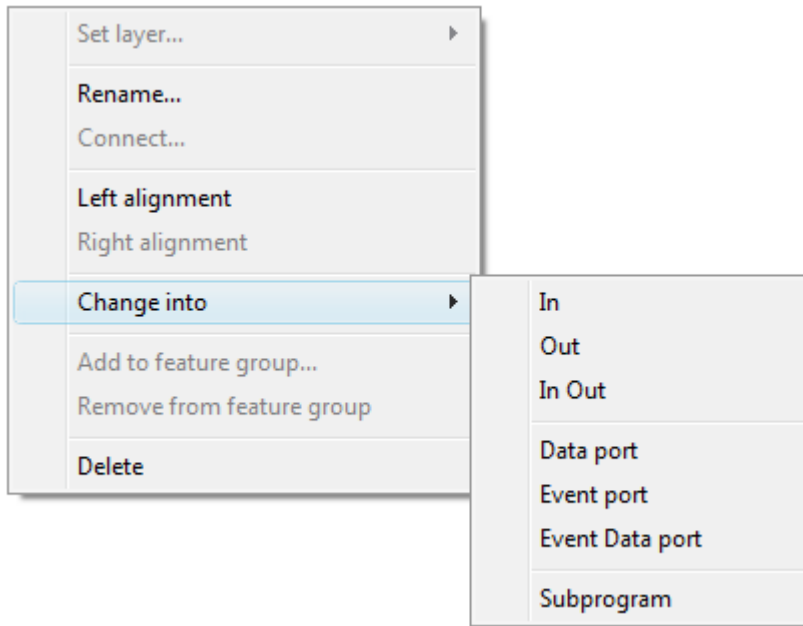
2.5.2. AADL package or component contextual menu



2.5.3. AADL sub package or subcomponent contextual menu



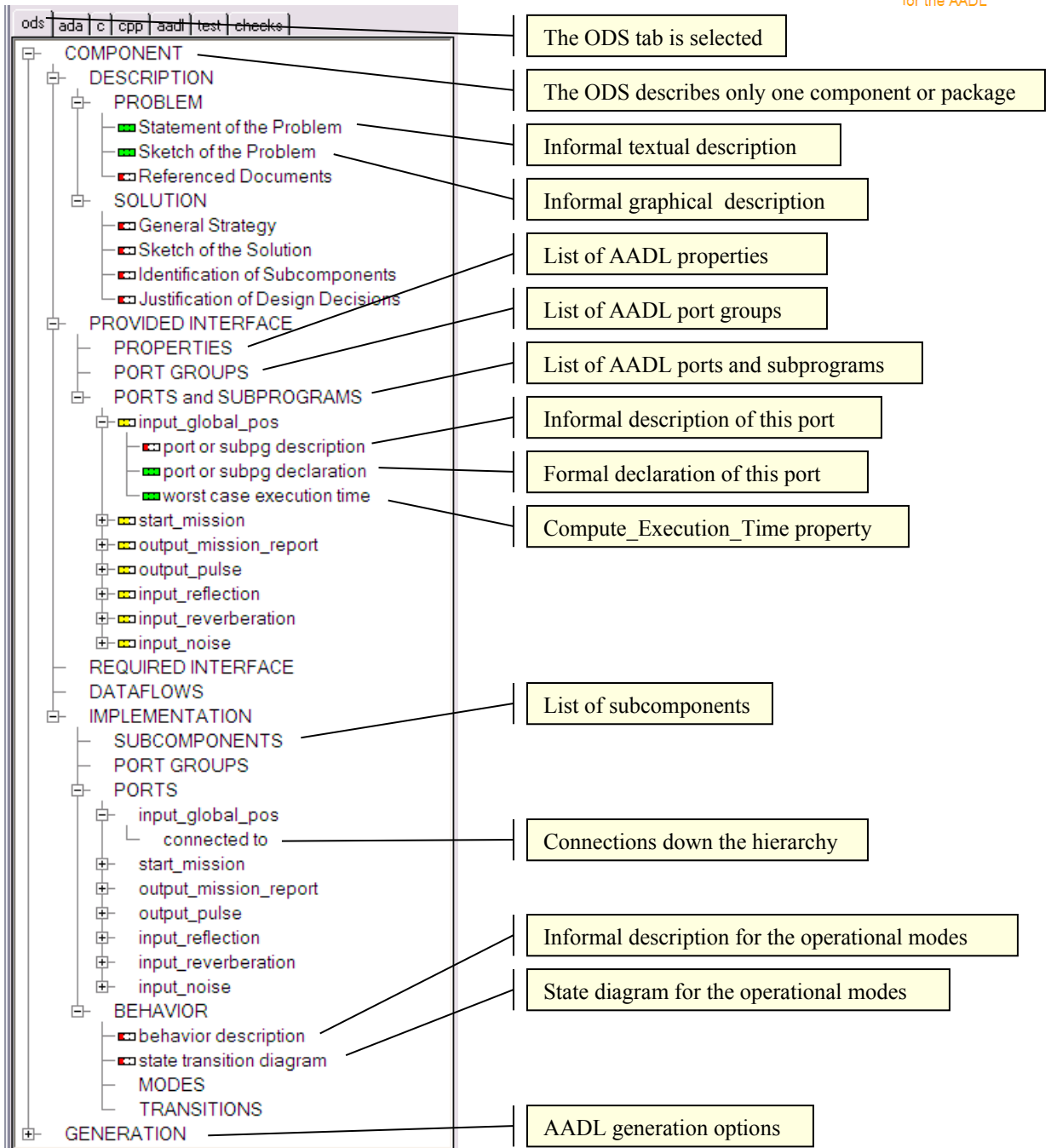
2.5.4. AADL feature contextual menu



2.6 *AADL textual editor*

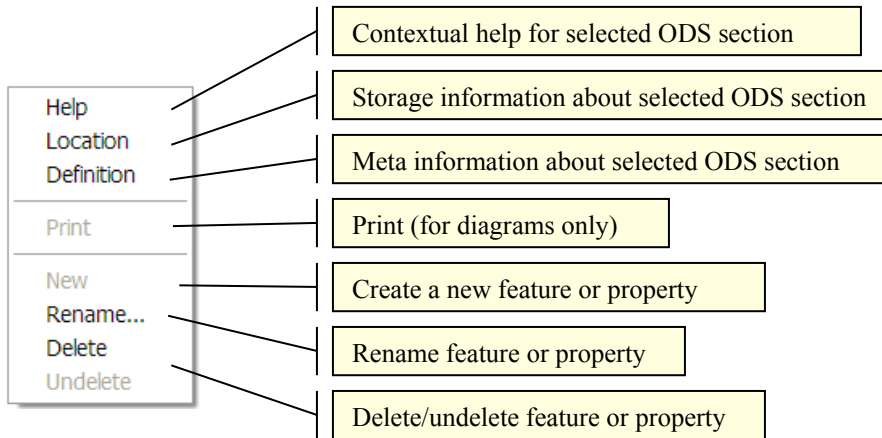
The **AADL** textual editor of **Stood** is a customized configuration of the generic structured design editor, called **ODS** editor in **Stood** terminology. The **ODS** is used to store in a well structured way, all the features and properties for each **AADL** component or package. The **ODS** can also hold additional design information like sketches and textual comments. The **ODS** is at the same time a guide for entering details into an **AADL** design model, and a frame for producing the **AADL** design documentation.

Next picture shows an example of the **ODS** of an **AADL** component. This list is displayed within the *component area* and is automatically updated when features or properties are created or deleted, and according to the category of the current component.

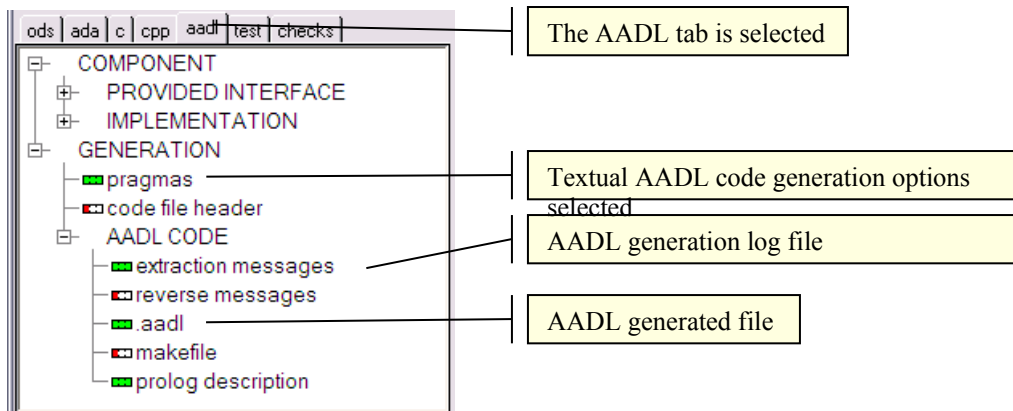


The colored gauge at the left side of an **ODS** section shows the completeness of the design model. A red gauge means that the corresponding section is still empty; a green gauge means that some information is available, and a yellow gauge for a non terminal section states that there are still empty subsections.

The **ODS** text offers the following contextual menu. Items of this menu may be greyed if they are not appropriate for the current selection in the list.



Several filters may be defined on the global **ODS** descriptor. Next picture shows the filtered view that is shown after textual **AADL** code has been generated. It is also possible to access this view at any time by choosing the *aadl* tab in the *property selector*.



3 Edition of AADL models

3.1 *Load existing models*

3.1.1. Open a project

Design models are related to a project. A given design model may belong to several projects, but it is always necessary to select and open a project before loading a design model. In **Stood** terminology, the current project is called the system configuration, often named system to make it short. Although there are many similarities, a **Stood** system doesn't always match an **AADL** system.

Use *File/Open project...* in the *main menu* to open an existing project. This opens a standard file navigator asking a **Stood** system configuration file to be selected. These files contain a list of references to design models, and are identified by a `.sync` file extension.

When a project has been properly loaded, its name is displayed in the *window title* and the list of the design models that are visible within this project is shown in the top left area of **Stood**. This area is called the *project area*. If the project is empty or if the design model references cannot be resolved, this area may be empty.

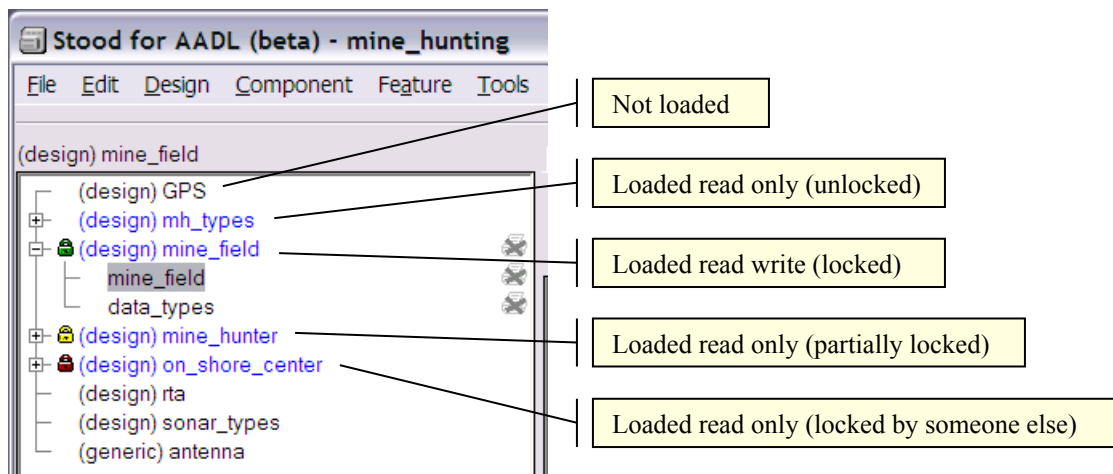
3.1.2. Load a design model

When a project is open, the design models it contains are not automatically loaded. When a design model is not loaded, its name is shown in *black* in the project view.

Perform a *single click* on the name of a design model to load it in a read only mode. This changes the colour of the name in the project view into *blue*, and initializes the other areas of the window with the contents of the loaded design model. However, no change will be permitted, and most menus and buttons will be inactive.

Perform a *double click* on the name of a design model, or use the *Lock* contextual menu or *Design/Lock design* in the *main menu* to load it in a read write mode. This has the same effect as a single click but a *green padlock* will be displayed at the left of the name in the *project area*, and changes will be allowed.

The *green padlock* means that the design model is locked for all the other concurrent sessions, thus providing a simple and efficient protection in a multi user environment. It may happen that a *red padlock* is displayed at the left of the name of a design model. This means that this model is currently in use by another user, and that it will not be possible to lock it until it is released. It may also happen that a *yellow padlock* is shown. This means that the design model is partially locked. In that case, a *green* or *red padlock* should be visible at a lower level in the components hierarchy.



3.2 Create new models

Any design model must be edited within a project. It is possible to create a new design model inside an existing project (refer to paragraph 3.1.1), otherwise it is necessary to firstly create a new project.

3.2.1. Create a new project

Use *File/New project...* in the *main menu* to create a new project. This opens a dialog box asking the name of the project to be entered. The result of this action will be to create a new **Stood** system configuration file with the given name and a `.sync` extension, in the default working directory.

This newly created project is empty by defaults. This means that the *project area* will be empty. It is necessary to either add existing design models to the project, or create new design models within this project.

3.2.2. Add or remove design models in a project

Use *File/Add to project...* (resp. *Remove from project*) in the *main menu* or the *Add...* (resp. *Remove*) *contextual menu* of the project view to let an existing design model be visible (resp. invisible) within the current project.

A newly added design model will be added to the list in the *project area*, but will not be automatically loaded. Please refer to paragraph 3.1.2 to know how to do to load a design model. A newly removed design model will be hidden in the *project area*, but will not be deleted.

3.2.3. Create a new design model

Use *Design/New design/aadl...* in the *main menu* or the *New/design... contextual menu* of the *project area* to create a new design model. This opens a dialog box asking the name of the design model to be entered. The result of this action will be to create a new **Stood** design subdirectory with the given name, in the same directory as the project. A directory can be recognized as a **Stood** design if it contains a file named `Stood.sto`.

The newly created design model will be added to the list in the *project area*, but will not be automatically loaded. Please refer to paragraph 3.1.2 to know how to do to load a design model.

3.2.4. Create a new design model from an existing AADL specification

It is possible to import an existing **AADL** specification into **Stood**. This **AADL** specification is transformed into a **Stood** design model thanks to an embedded textual **AADL** 1.0 syntactic analyser and a set of semantic transformation rules.

Use *Design/New design from/aadl...* in the *main menu* or *New from/aadl... contextual menu* of the *project area* to create a new design model from an existing **AADL** specification. This opens a standard file navigator asking a textual **AADL** file to be selected. Files are recognized to be textual **AADL** files if they have a `.aadl` extension.

The result of this action will be to analyse all the textual **AADL** files located within the same directory as the selected file, and to create a design model having the same name as the selected file.

The newly created design model will be added to the list in the *project area*, but will not be automatically loaded. Please refer to paragraph 3.1.2 to know how to do to load a design model.

Note that **Stood** makes the assumption that the selected file has the same name as the root of the **AADL** component or package hierarchy to be imported. When the textual **AADL** source contains several **AADL** hierarchies, it is necessary to create several design models, after having either selected the appropriate file in the source directory or properly renamed the unique input file.

3.2.5. Create a new design from existing Ada or C source files

It is possible to reverse engineer existing **Ada** or **C** source code into **Stood**. This legacy code is transformed into a **Stood** design model thanks to an embedded **Ada** and **C** syntactic analyser and a set of semantic transformation rules.

Use *Design/New design from/ada...* (resp. *c...*) in the *main menu* or *New from/ada...* (resp. *c...*) *contextual menu* of the *project area* to create a new design model from legacy **Ada** (resp. **C**) code. This opens a standard file navigator asking an **Ada** (resp. **C**) file to be selected. Files are recognized to be **Ada** (resp. **C**) files if they have a `.ads` or `.adb` (resp. `.h` or `.c`) extension. The result of this action will be to analyse all the source files located within the same directory as the selected file, and to create a design model having the same name as the selected file.

Note that **Stood** makes the assumption that the selected file has the same name as the main source code file to be imported. The newly created design model will be added to the list in the *project area*, but will not be automatically loaded. Please refer to paragraph 3.1.2 to know how to do to load a design model.

3.3 AADL packages

To create a new **AADL** package, use *Design/New design/aadl package* in the *main menu*. A package represents a library of reusable components. Unlike within a process, these

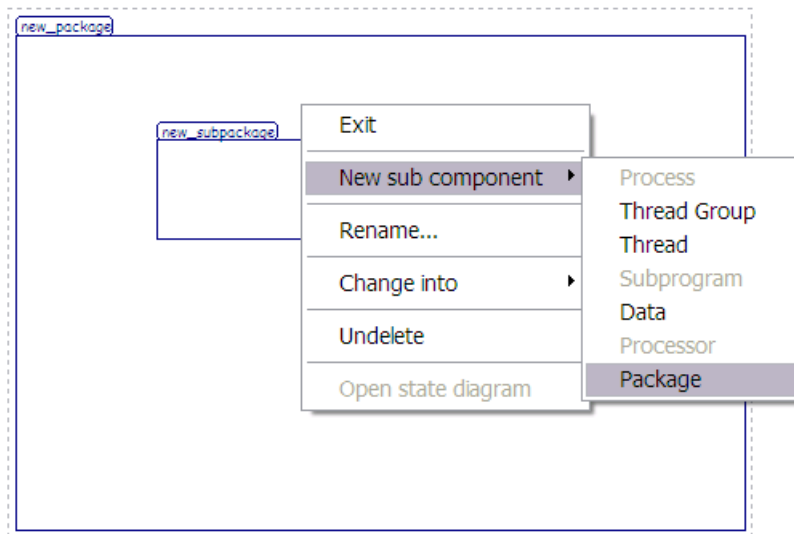
components are not instantiated. If no component has been defined inside a package, then a dummy data component type will be generated to comply with the **AADL** syntax:

```
PACKAGE empty_package
PUBLIC

  DATA void
  END void;

END empty_package;
```

Although the **AADL** specifies a flat representation of package hierarchies, a sub package will be graphically represented as being contained by its parent package:



```
PACKAGE new_package
PUBLIC

  DATA void
  END void;

END new_package;

PACKAGE new_package::new_subpackage
PUBLIC

  DATA void
```

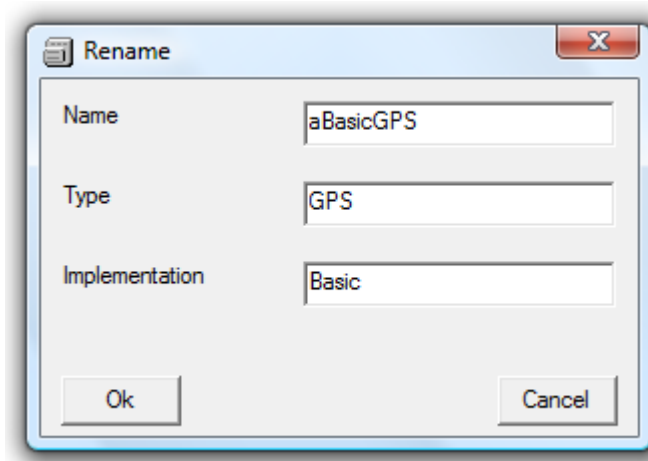
```
END void;
```

```
END new_package::new_subpackage;
```

3.4 AADL components

In **Stood**, **AADL** components represent in effect subcomponents that are instances of component types or implementations.

During **AADL** code generation, component type name will be set by default to subcomponent name, and component implementation name will be set to `others`. It is of course possible to change these default names so that several subcomponents can share a same type or implementation. The *renames* dialog box can be used for this purpose.



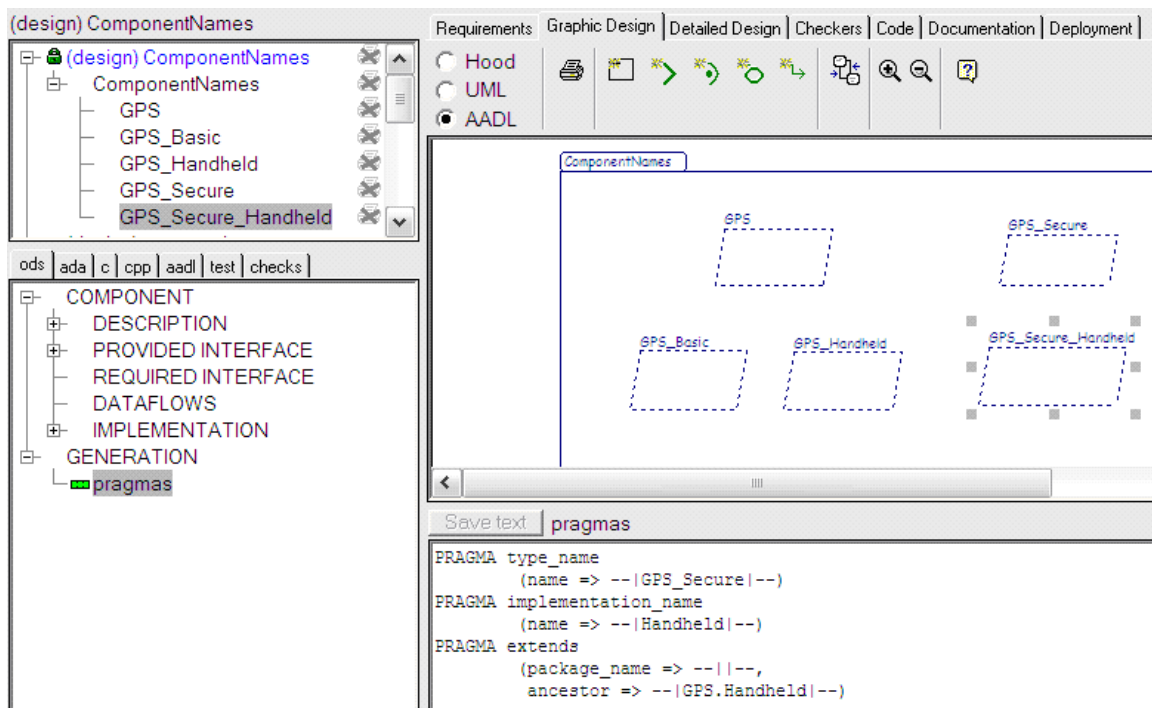
Component extension is not shown during the design process in most cases. The **AADL** code generation pragma *extend* must be used to specify that the specified component holds a type or implementation extension. Refer to chapter 4.2 to get more details about these **AADL** code generation pragmas.

Note that it is possible to describe graphically data component extensions when they are declared in a package and with the help of the **UML** editor (inheritance link)

example:

In the design model below, following AADL generation pragmas have been set:

Subcomponent	<i>type_name</i>	<i>implementation_name</i>	<i>extends</i>
GPS	-	-	-
GPS_Basic	GPS	Basic	-
GPS_Handheld	GPS	Handheld	GPS.Basic
GPS_Secure	-	-	GPS
GPS_Secure_Handheld	GPS_Secure	Handheld	GPS.Handheld



The screenshot shows the AADL design tool interface. On the left, a tree view displays the component hierarchy: (design) ComponentNames, ComponentNames, GPS, GPS_Basic, GPS_Handheld, GPS_Secure, and GPS_Secure_Handheld. Below the tree, a list of generation options is shown, with 'pragmas' selected. The main workspace displays a diagram of the component hierarchy with dashed boxes representing components: GPS, GPS_Secure, GPS_Basic, GPS_Handheld, and GPS_Secure_Handheld. The bottom panel shows the generated AADL code for the 'pragmas' section.

```

Save text | pragmas
PRAGMA type_name
  (name => --|GPS_Secure|--)
PRAGMA implementation_name
  (name => --|Handheld|--)
PRAGMA extends
  (package_name => --|--,
   ancestor => --|GPS.Handheld|--)
  
```

The corresponding textual AADL code that is generated is:

```

THREAD GPS
END GPS;
  
```

```

THREAD IMPLEMENTATION GPS.Basic
END GPS.Basic;
  
```

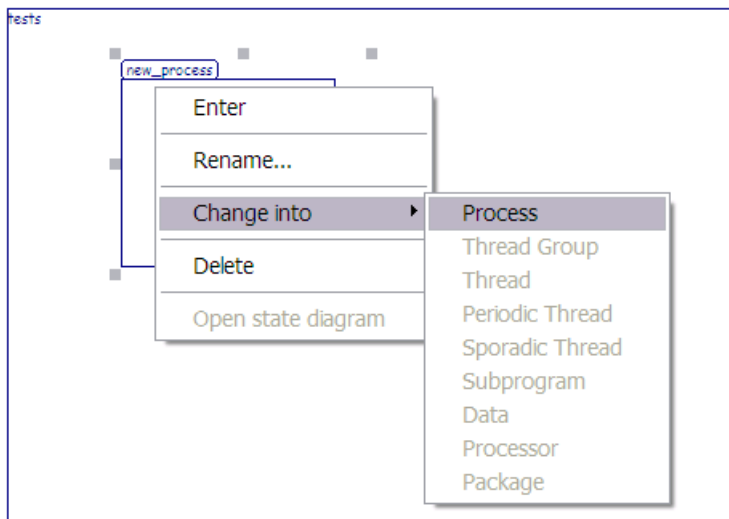
```
THREAD IMPLEMENTATION GPS.Handheld EXTENDS GPS.Basic  
END GPS.Handheld;
```

```
THREAD GPS_Secure EXTENDS GPS  
END GPS_Secure;
```

```
THREAD IMPLEMENTATION GPS_Secure.Handheld EXTENDS GPS.Handheld  
END GPS_Secure.Handheld;
```

3.4.1. AADL Processes

To create an **AADL** process instance, use *Design/New design/aadl process* in the *main menu*.



The **AADL** code that is generated for an empty process is as follow:

```
SYSTEM tests  
END tests;
```

```
SYSTEM IMPLEMENTATION tests.others  
SUBCOMPONENTS  
    new_process : PROCESS new_process;  
END tests.others;
```

```
PROCESS new_process
PROPERTIES
  Stood::Box_Position => "(X1 => 222,Y1 => 101,X2 => 636,Y2 => 501)";
END new_process;
```

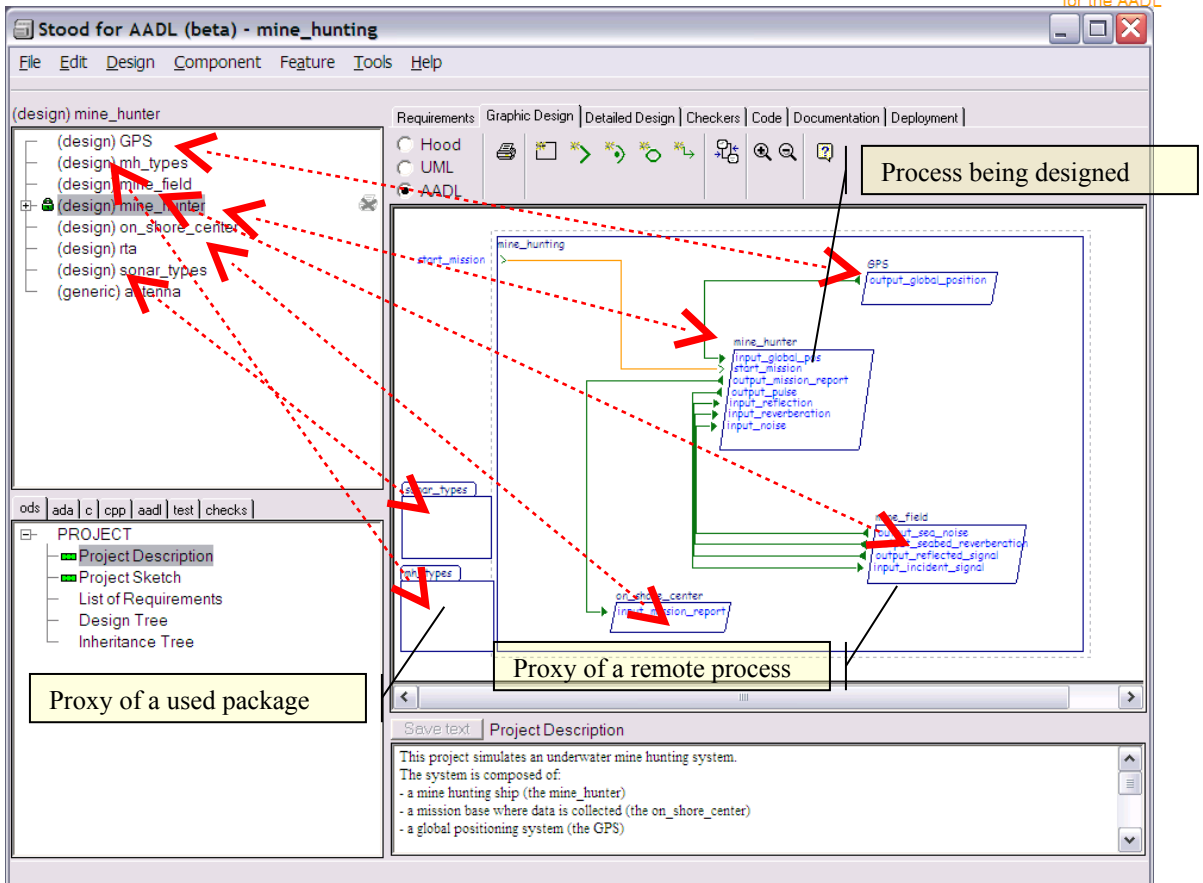
Note that an **AADL** process is necessarily represented by a **Stood** design that is a root of a hierarchy of software components that will be transformed into an executable application or a linkable library, at the end of the software development process. If the project contains several processes, it is thus mandatory to create several design models.

It is possible to represent the interface of other remote processes or used packages in the enclosing system. These components or packages act as proxy of the actual processes or packages that must correspond to other design models of the same project.

To create these proxies (also called environment components in **Stood** terminology), create sibling components to the main process, and give them the name of an existing design model of the project.

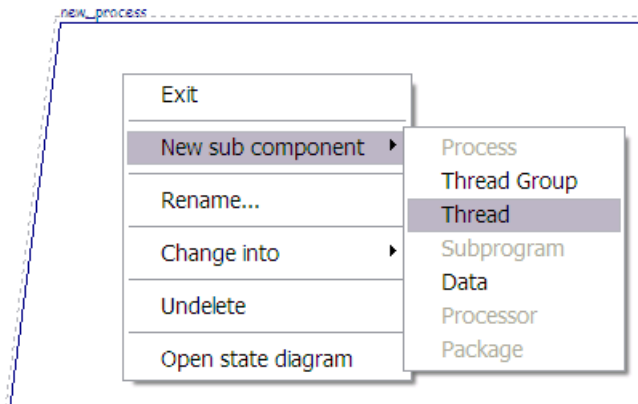
example:

The `mine_hunting` project is composed of 8 design models. In the current session, process `mine_hunter` is being designed, and is locked. A proxy of processes `GPS`, `mine_field` and `on_shore_center` is used to show the interaction of the 4 processes within the system associated to the project. Additionally, 2 packages `mh_types` and `sonar_types` are made visible, so that the various data types they are exporting will be recognized as port or subprogram parameter classifiers. Note that 2 other design models are not shown graphically, because their role is different: `antenna` is a generic component that is intanciated at a lower level in the design hierarchy, whereas `rta` represents an library package used during **Ada** code generation.

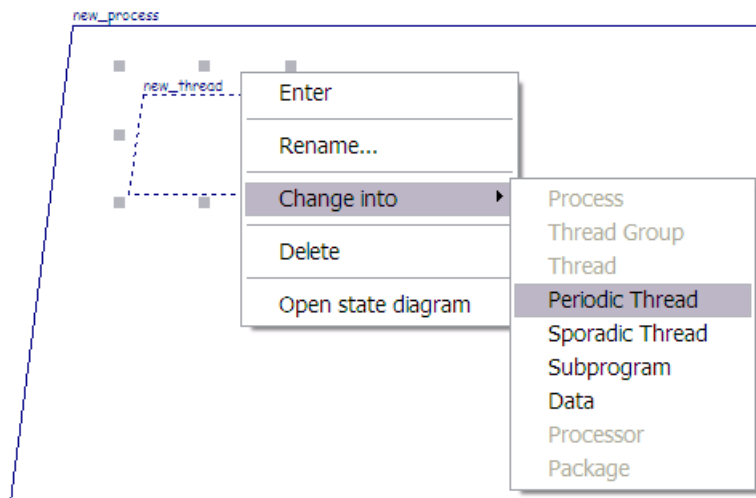


3.4.2. AADL Threads

To create a new thread in a process or a thread group, use the *new AADL component* button in the *tool bar*, or the *New sub component* contextual menu. To create a new thread in a package, only use the *New sub component* contextual menu, as the button always creates a data component.



Threads are created with aperiodic dispatch protocol. It is however possible to specify a periodic or sporadic dispatch protocol with the *Change into ...* contextual menu.



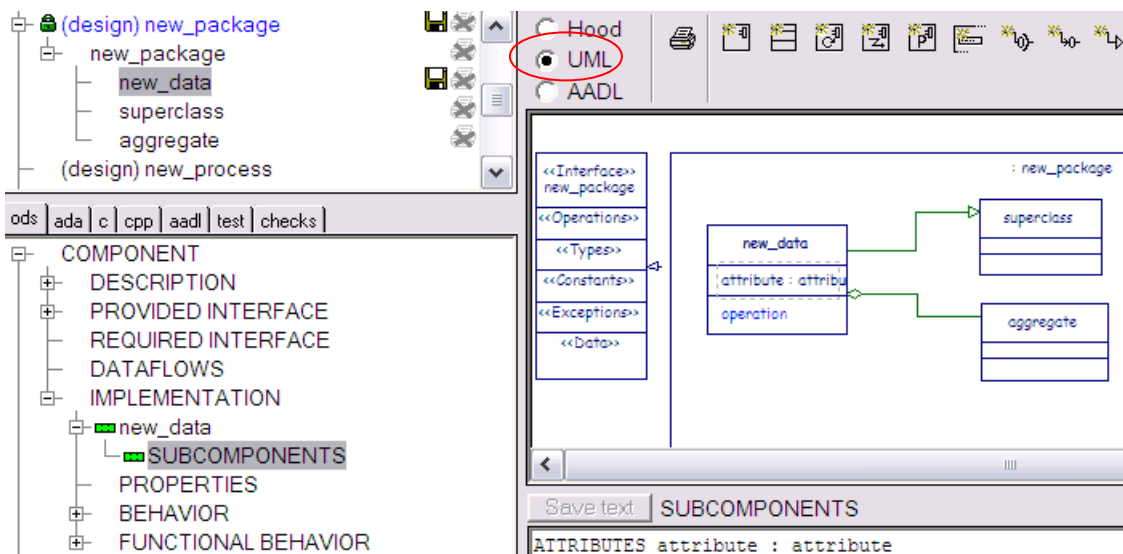
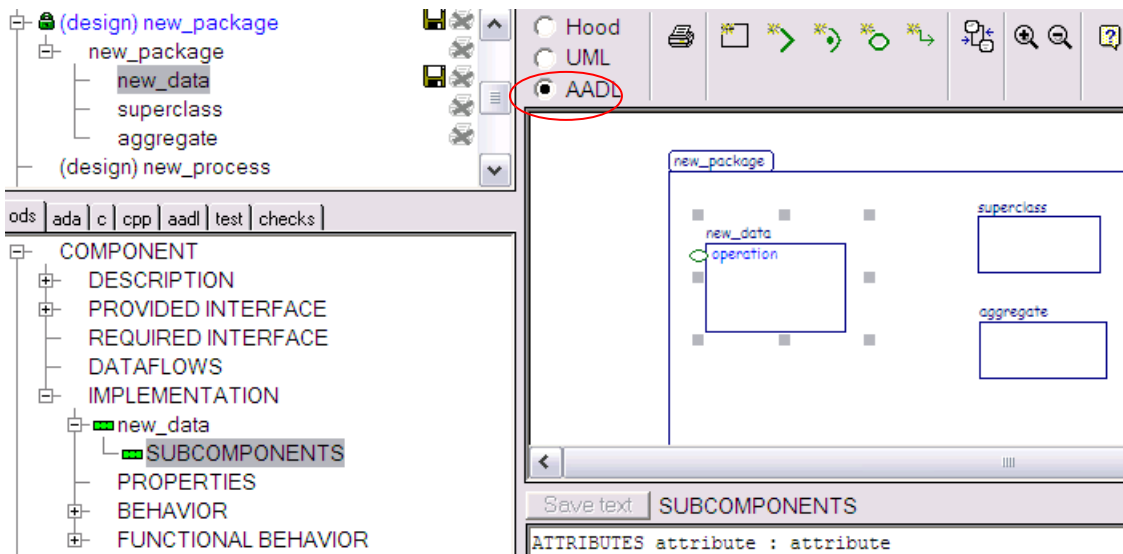
3.4.3. AADL Thread groups

To create a new thread group in a process, a thread group or a package, use the *New sub component* contextual menu.

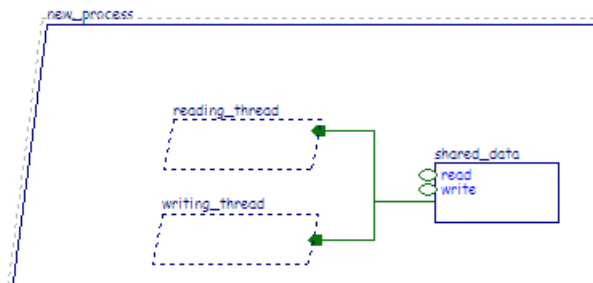
Stood also allows temporarily defining a thread subcomponent within a thread component. The container thread will then be automatically changed into a thread group.

3.4.4. AADL Data

Data components created within a package are mapped to a class in the **Stood** model. It will be sometimes more appropriate to use the **UML** editor to better show class structure (attributes and operations) and their relationships (inheritance, aggregation). Next pictures show both views for the same model:



On the contrary, data components that are created as subcomponents of a process, thread group or thread are mapped to a shared protected object in the **Stood** model. Use of data access connections in the **AADL** graphical notation will show the actual data sharing:



3.4.5. AADL Subprogram components

In most cases, subprogram components will be automatically created during the **AADL** code generation, from the definition of subprogram features.

example:

AADL generated code from the diagram above will contain the definition of the two subprogram components `read` and `write`, although they don't appear in the diagram.

```

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

SUBPROGRAM read
END read;

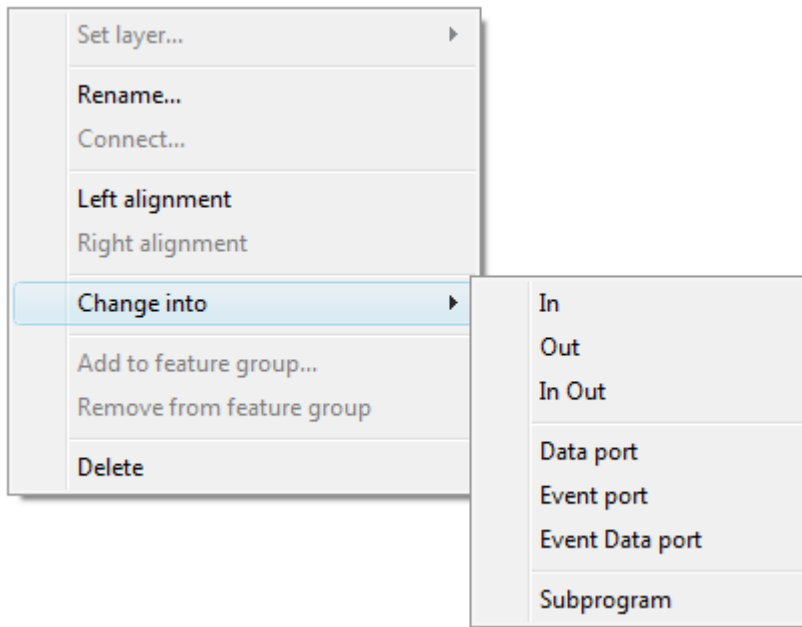
SUBPROGRAM write
END write;

```

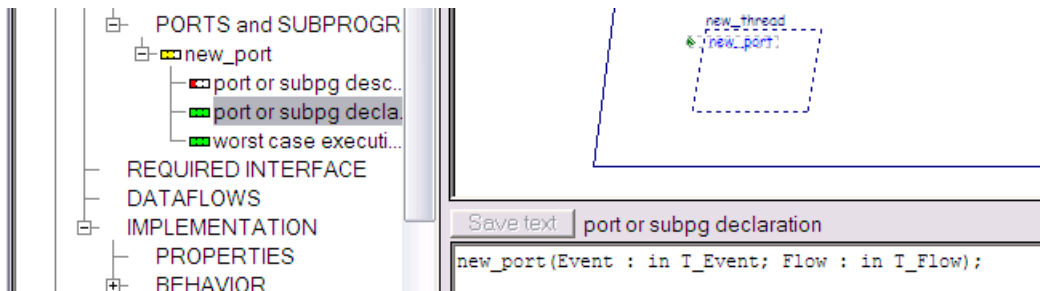
3.5 AADL features

3.5.1. AADL Ports

To create a port, use the *new port* button in the *tool bar*. New ports are created as in event by default. Use the *Change into ... contextual menu* to change the port kind and direction:



When a port is selected in the graphical editor, its formal declaration is shown in the text input area:

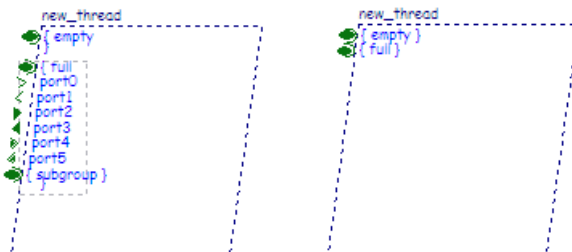


In a **Stood** design model, **AADL** ports are stored as operations with parameters. An event or event data port has an `Event` parameter. A data port or event data port has a parameter which name must be other than `Event`. Default name for a data parameter is `Flow`, and may be changed, as well as its parameter type which is set to `T_Flow` by default. Removing or renaming the `Event` parameter will remove the event nature of the port.

To validate a change in the port declaration section, it is mandatory to use the *Save text* button, or *contextual menu* or the *Ctrl-S* keyboard shortcut.

3.5.2. AADL Feature groups

To create a new feature group, use the corresponding button of the tool bar. It is then possible to drag new ports or other feature groups inside the two enclosing brackets delimitating the feature group. A feature group may be open or close as shown in the pictures below:

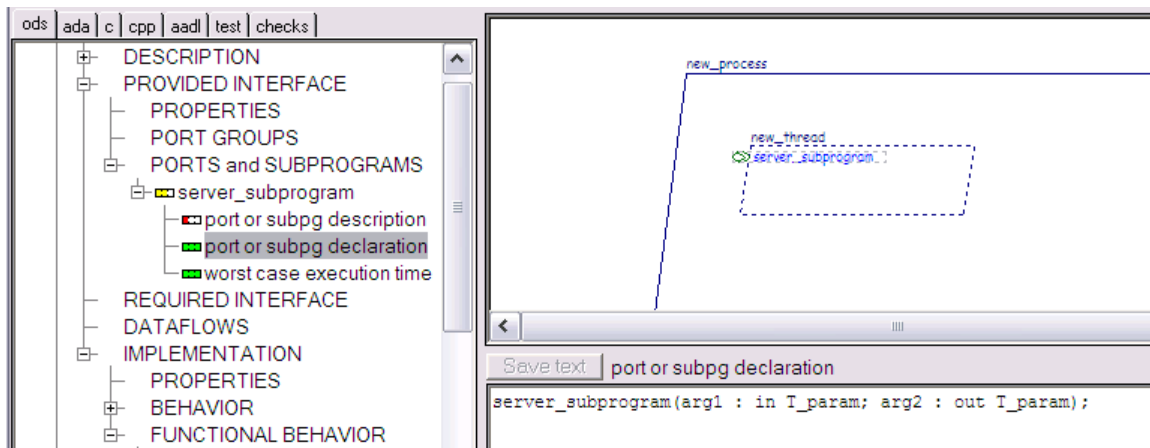


To open a closed feature group, double click on its name or use *enter* contextual menu. To close an opened feature group, double click on its name or use *exit* contextual menu.

To include a feature to a feature group, drag it inside the opened port group or use *add to feature group* contextual menu. To remove a feature from a feature group, drag it outside the opened feature group or use *remove from feature group* contextual menu.

3.5.3. AADL Subprogram features

To create a new subprogram feature, use the corresponding button of the tool bar. A new subprogram is created without any parameter by default. To edit the parameters list of a subprogram feature, use the *port or subpg declaration* section of the **ODS**.



To validate a change in the subprogram declaration section, it is mandatory to use the *Save text* button, or *contextual menu* or the *Ctrl-S* keyboard shortcut.

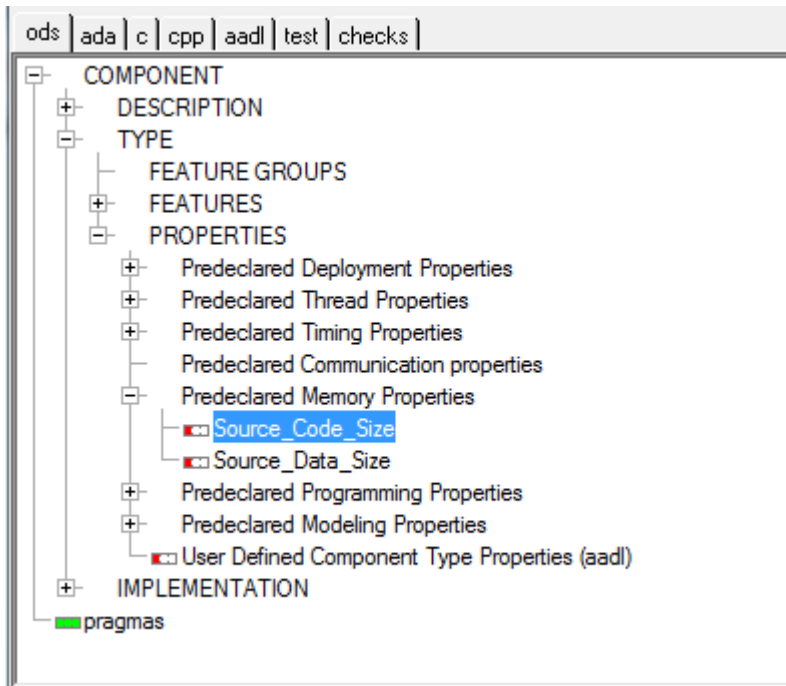
3.6 AADL connections

The new connection button of the tool bar must be used to create:

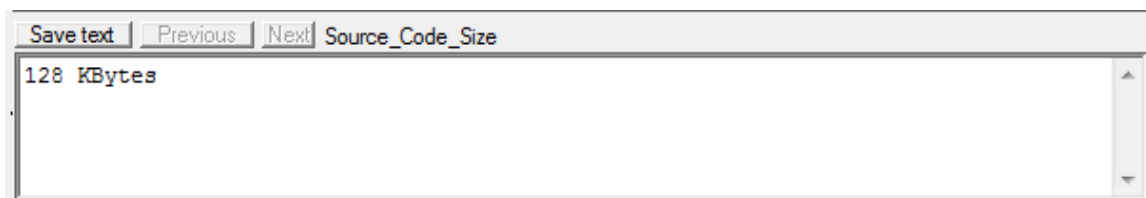
- a connection between two ports
- a connection between two feature groups
- an access connection between a data component and a thread component

3.7 AADL properties

The list of all the predefined AADL properties is included into the ODS. To insert the property value for the selected component or feature, select the corresponding property name in the list and write the value in the text editing area.



To validate a change in the property section, it is mandatory to use the *Save text* button, or *contextual menu* or the *Ctrl-S* keyboard shortcut.



3.7.1. Stood property set

In order to be able to propagate graphical information through **AADL** specifications, **Stood AADL** code generator automatically introduces a few specific properties. Generation of these properties may be avoided by using the pragma *no_graphics*.

```
property set Stood is
  Box_Position : aadlstring
    applies to (system, data, subprogram, thread, thread group,
```

```

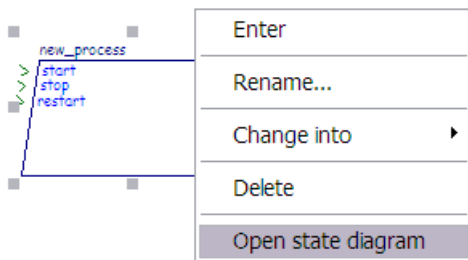
        process, memory, processor, bus, device);
    Link_Position : aadlstring
        applies to (connections);
end Stood;

```

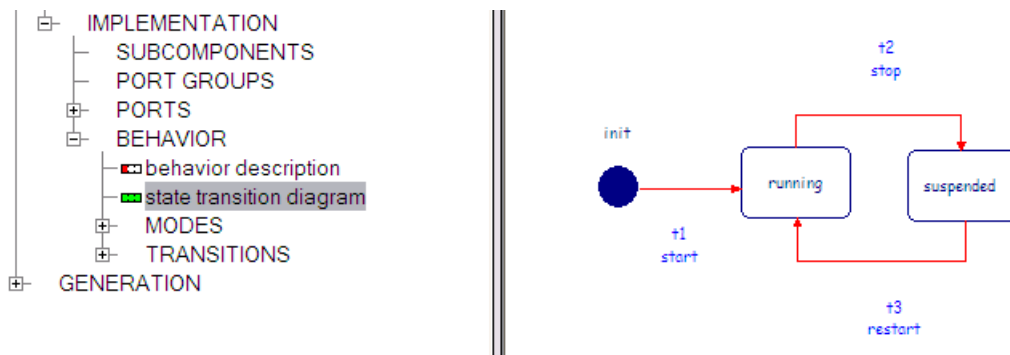
3.8 AADL modes

Stood offers an incomplete support for **AADL** modes. It is possible graphically define the operational modes of a process thanks to a *state transition diagram*. However, There is currently no support of the *in modes* statements.

Use the *mode diagram* button of the tool bar or choose *open state diagram* in the contextual menu when the main process component is selected in the **AADL** diagram. The process must provide one or several event ports that may be used as transition triggers.



When the *state transition diagram* is open, a specific tool bar is available to create an initial mode, standard modes and transitions. When a transition is selected, use the *select transition event* button to associate one of the provided event ports of the process.



Following **AADL** code will be generated from this model:

```

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

PROCESS IMPLEMENTATION new_process.others
SUBCOMPONENTS
  new_thread : THREAD new_thread;
MODES
  init : INITIAL MODE;
  running : MODE;
  suspended : MODE;
  init -[ start ]-> running;
  running -[ stop ]-> suspended;
  suspended -[ restart ]-> running;
END new_process.others;

```

3.9 *AADL flows*

Stood can be used to declare flow specifications. There is however currently no support for flow implementations and end to end flows. **Stood** represents internally **AADL** data ports as operations with parameters. The name of the operation is used for the port name; if not `Event`, the type of the parameter is used for the data port classifier; and the name of the parameter is used to specify flows. Default value for a data port operation parameter is `Flow`. There will be no flow specification generated for default values of this parameter, however, if this parameter name is changed to the name of a flow, then

corresponding flow specifications will be generated in terms of *flow paths*, *flow sources* and *flow sinks*.

example:

The various ports that are shown in the example below have the following **Stood port or subpg declaration** in their **ODS** section:

In process new_process:

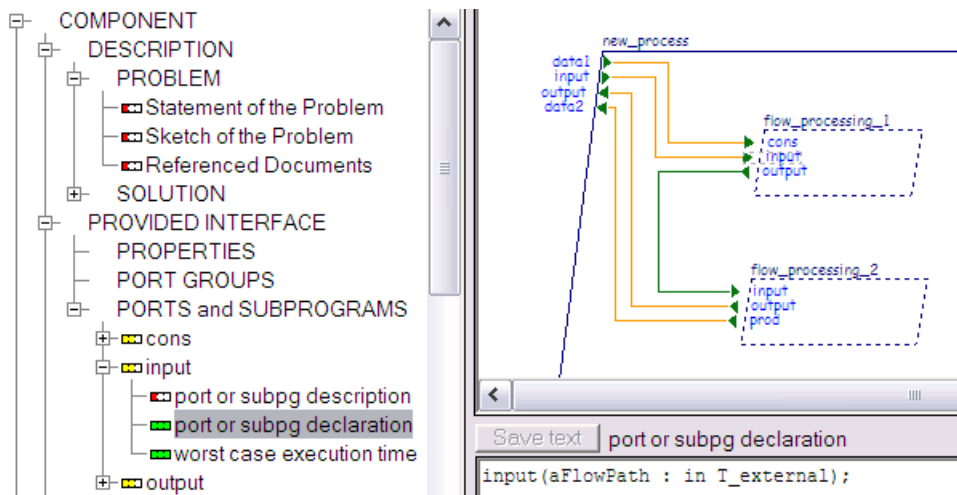
```
data1(aFlowSink : in T_Flow);
input(aFlowPath : in T_external);
output(aFlowPath : out T_external);
data2(aFlowSource : out T_Flow);
```

In thread flow_processing_1:

```
cons(aFlowSink : in T_Flow);
input(aFlowPath : in T_external);
output(aFlowPath : out T_internal);
```

In thread flow_processing_2:

```
input(aFlowPath : in T_internal);
output(aFlowPath : out T_external);
prod(aFlowSource : out T_Flow);
```



The **AADL** code that is generated is as follow:

```
PROCESS new_process
FEATURES
  data1 : IN DATA PORT T_Flow;
  input : IN DATA PORT T_external;
  output : OUT DATA PORT T_external;
  data2 : OUT DATA PORT T_Flow;
FLOWS
  aFlowPath : FLOW PATH input -> output;
  aFlowSource : FLOW SOURCE data2;
  aFlowSink : FLOW SINK data1;
END new_process;

PROCESS IMPLEMENTATION new_process.others
SUBCOMPONENTS
  flow_processing_1 : THREAD flow_processing_1;
  flow_processing_2 : THREAD flow_processing_2;
CONNECTIONS
  PORT data1 -> flow_processing_1.cons;
  PORT input -> flow_processing_1.input;
  PORT flow_processing_2.output -> output;
  PORT flow_processing_2.prod -> data2;
  PORT flow_processing_1.output -> flow_processing_2.input;
PROPERTIES
  Stood::Box_Position => "361 212 719 456";
END new_process.others;

THREAD flow_processing_1
FEATURES
  cons : IN DATA PORT T_Flow;
  input : IN DATA PORT T_external;
  output : OUT DATA PORT T_internal;
FLOWS
  aFlowPath : FLOW PATH input -> output;
  aFlowSink : FLOW SINK cons;
PROPERTIES
  Dispatch_Protocol => Aperiodic;
  Stood::Box_Position => "398 143 682 295";
END flow_processing_1;

THREAD flow_processing_2
FEATURES
  input : IN DATA PORT T_internal;
  output : OUT DATA PORT T_external;
  prod : OUT DATA PORT T_Flow;
FLOWS
```

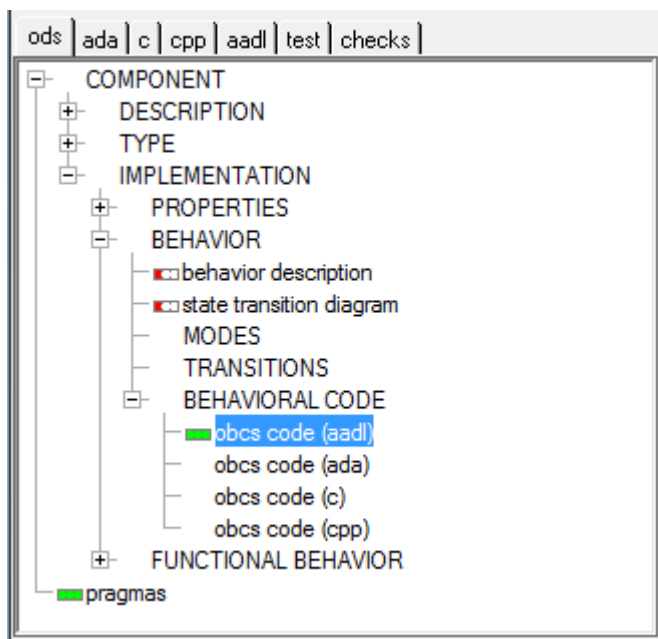
```

aFlowPath : FLOW PATH input -> output;
aFlowSource : FLOW SOURCE prod;
PROPERTIES
Dispatch_Protocol => Aperiodic;
Stood::Box_Position => "373 410 689 563";
END flow_processing_2;

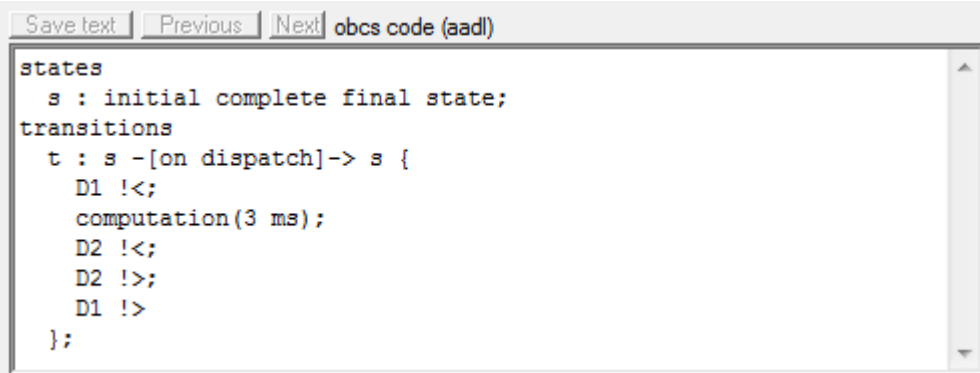
```

3.10 Behavior Annex

The **ODS** contains sections that can be used to insert **AADL** Behavior Annex code for Thread and Subprogram components.



To validate a change in a Behavior Annex section, it is mandatory to use the *Save text* button, or *contextual menu* or the *Ctrl-S* keyboard shortcut.



```

Save text Previous Next obcs code (aadl)
states
  s : initial complete final state;
transitions
  t : s -[on dispatch]-> s {
    D1 !<;
    computation(3 ms);
    D2 !<;
    D2 !>;
    D1 !>
  };

```

The AADL code that is generated is as follow:

```

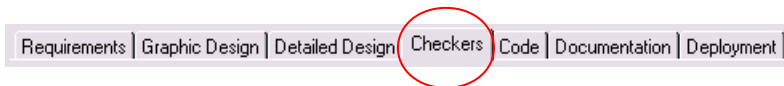
THREAD IMPLEMENTATION T.i1
PROPERTIES
  Dispatch_Protocol => Periodic;
  Compute_Execution_Time => 5ms..5ms;
  Period => 15 ms;
  Stood::Box_Position => "266 333 449 516";
ANNEX Behavior_Specification {**
  states
    s : initial complete final state;
  transitions
    t : s -[on dispatch]-> s {
      D1 !<;
      computation(3 ms);
      D2 !<;
      D2 !>;
      D1 !>
    };
  **};
END T.i1;

```

4 Processing of AADL models

4.1 *Generate design verification reports*

To enter the design verification mode, select the *Checkers* tab in the *life cycle selector*:

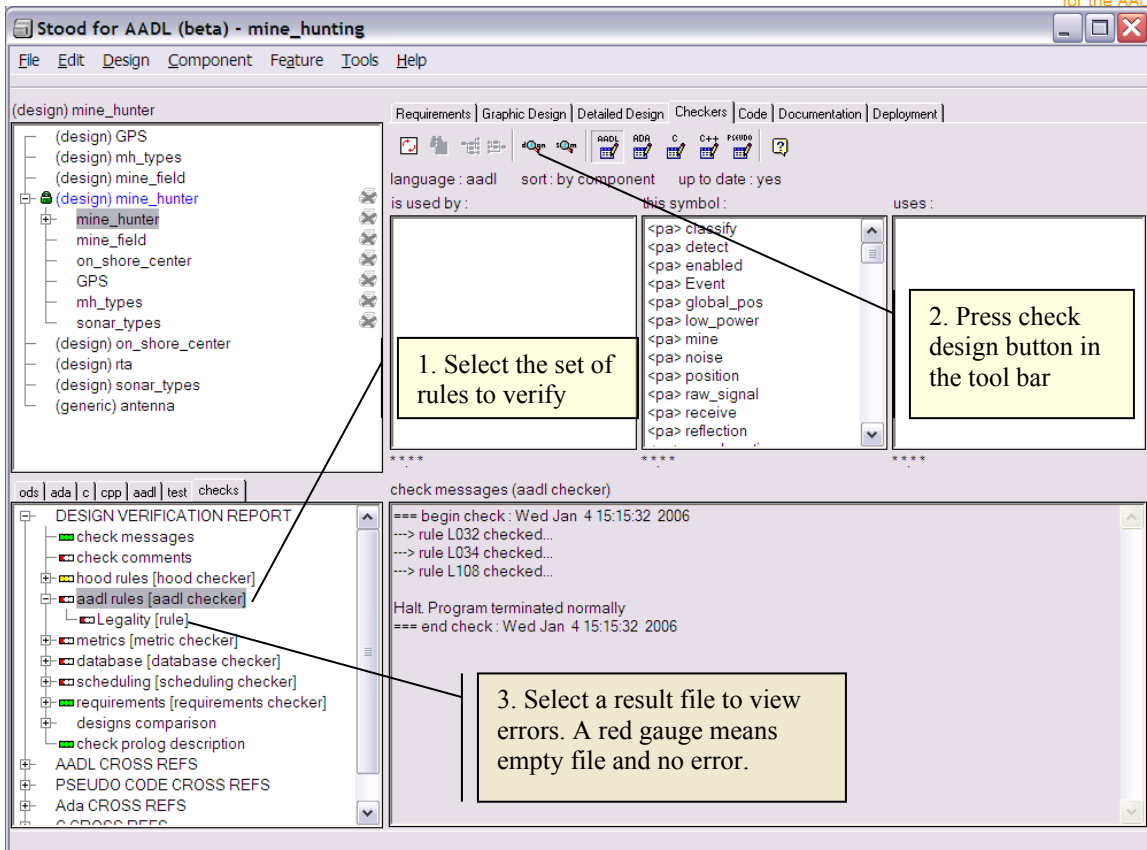


Stood includes an internal cross reference table and several embedded design model verification tools:

- requirements traceability matrix
- schedulability analysis (basic test only)
- design architecture metrics
- **HOOD** rules compliancy
- **AADL** legality rules compliancy (under development)

To activate one of these verification tools, perform the following sequence of actions:

- *step1*: select the set of rules to verify in the component area
- *step2*: click on the check design button in the tool bar
- *step3*: select a result file in the component area. If a red gauge is shown, this means that the result file is empty and that there is no error.

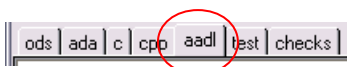


4.2 Generate textual AADL code

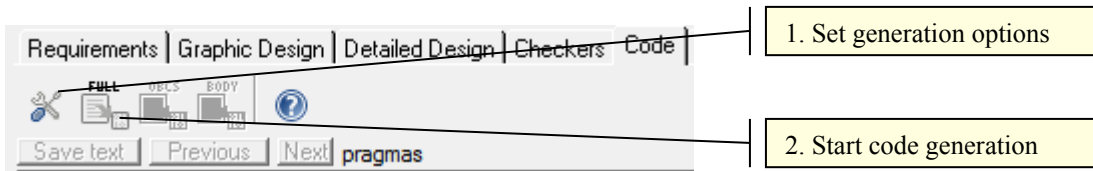
Textual AADL code can be generated at any time from the current design model. To activate the AADL code generator, switch the *life cycle selector* to the *code* tab.



If `aadl` has been specified as the default target language, then this will automatically open the AADL code generator. Else, it may be necessary to select the *aadl* tab in the *property selector*.



The first view of the code sub window gives access to generation options, called pragmas in **Stood** terminology and that are described below. To set an option, first select the component on which it should apply, then click on the *add pragma* button, and select the appropriate pragma in the list. When a pragma is set, its name is preceded by a >> tick. The list of all the currently set pragmas is shown in the editing area where it is possible to remove or duplicate them, and change the value of their arguments. Supported pragmas for the **AADL** code generator are listed in the next sections. If no pragma is set, default code generation rules will be applied.

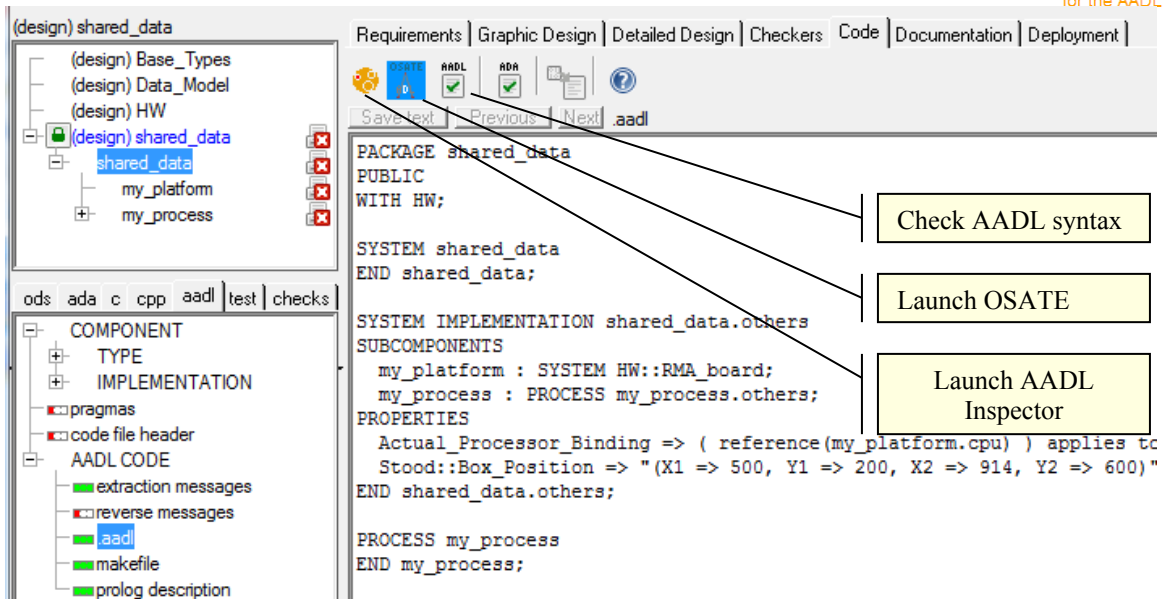


To start the generation of the textual **AADL** code, select *Tools/Code/Full extraction* in the main menu or simply press *full extraction* button in the tool bar, then select *OK* in the dialog box. When completed, the result of the code generation process is shown in another view of the code sub window, showing the result of the generation process. By default, this view shows generation log messages if the root component is selected in the *project area*.

For each package or component of the design model, select the *.aadl* section in the *component area* to edit the corresponding generated **AADL** code. If the pragma *one_file_per_component* was not set, then the whole **AADL** specification will be shown when the root component or package is selected.

The **AADL** syntax can be checked with the *check aadl* button of the tool bar. In addition, it is possible to launch external tools that will process or analyse the textual **AADL** specification that has been generated. For instance, it is possible to use **OSATE** to transform textual **AADL** generated by **Stood** into **XML AADL** files. Note that the analysis tool **AADL Inspector** is automatically launched after **AADL** code generation.

It is possible to make changes in these source files using the *text input area*. Changes must be saved with *Save text* button or *contextual menu*, or the *Ctrl-S* keyboard shortcut.



In addition, if the pragma *reverse* was set, any changes in the generated source files that are done between the round trip engineering tags will be fed back to the design model thanks to the round-trip engineering feature of **Stood**. To activate this feature, you must press the *reverse* button of the tool bar. Note that this feature is currently active for Behavior Annex behavioural sections only (refer to chapter 3.10).

The **AADL** source files can also be edited directly from the file system. Default location of generated code is the `_aadl` subdirectory in the directory of the design. To open it from **Stood**, use *Tools/Open directory/Design directory* in the main menu.

4.3 Generate Ada source code

Ada source code can be generated at any time from the current design model. To activate the **Ada** code generator, switch the *life cycle selector* to the *code* tab.



If `ada` has been specified as the default target language, then this will automatically open the **Ada** code generator. Else, it may be necessary to select the *ada* tab in the *property*

selector.



4.4 *Generate C source code*

C source code can be generated at any time from the current design model. To activate the C code generator, switch the *life cycle selector* to the *code* tab.



If *c* has been specified as the default target language, then this will automatically open the C code generator. Else, it may be necessary to select the *c* tab in the *property selector*.



4.5 *Generate design documentation*

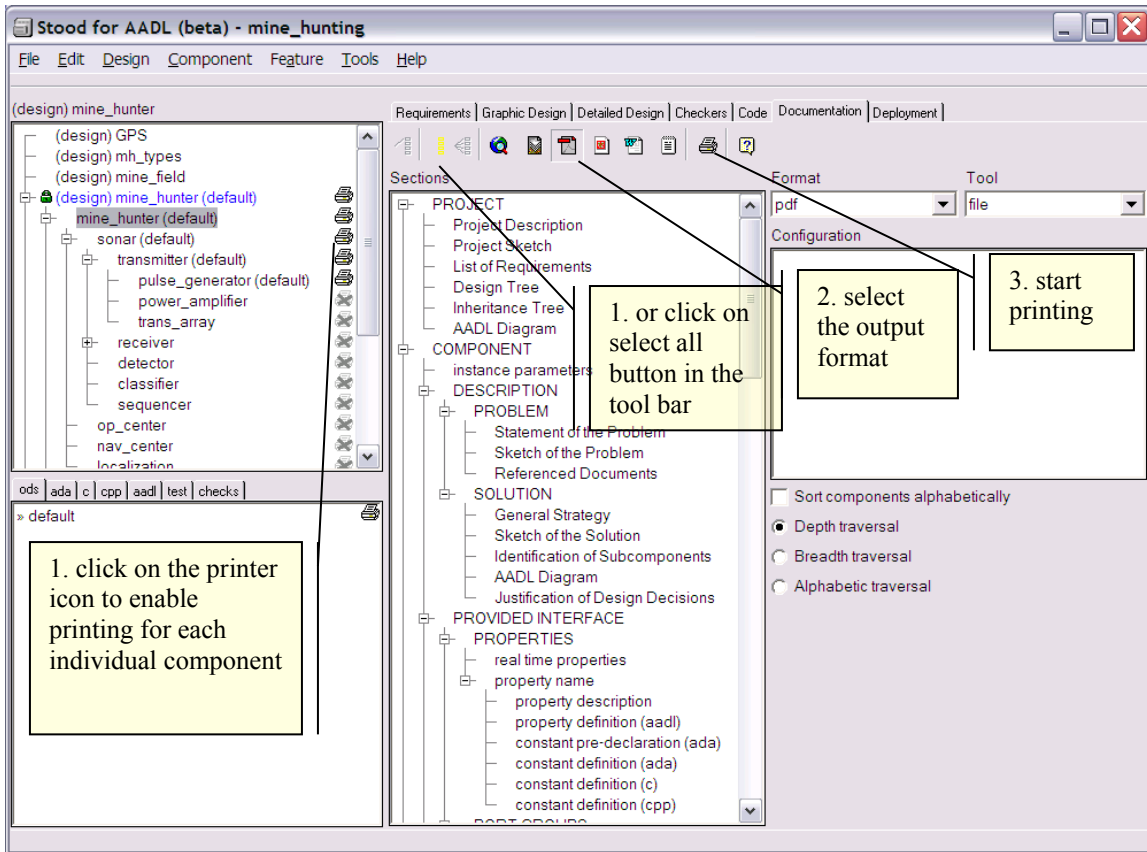
To enter the documentation production mode, select the *documentation* tab in the *life cycle selector*:



To create a full printable document from the current design model, operate as follow:

- *step1*: select the components to be printed, or select all with the appropriate button
- *step2*: select the output format among **HTML**, **MIF** (for FrameMaker™), **PDF**, **PostScript**, **RTF** (for Word™) or **ODT** (For OpenOffice).
- *step3*: click on the *print* button.

This opens a standard file dialog asking an output file name to be entered. Default location for generated documentation is the `_doc` subdirectory of the current design model directory. The current design directory can be opened from **Stood** by choosing *Tools/Open directory/design directory* in the *main menu*.





www.ellidiss.com
stood@ellidiss.com

Ellidiss Software
Triad House
Mountbatten Court
Worall Street
Congleton
Cheshire
CW12 1DT
UK

+44 1260 291 449

Ellidiss Technologies

24 quai de la douane
29200 Brest
Brittany
France

+33 298 451 870



www.aadl.info