TASTE Tutorial
v1.1

Maxime Perrotin
Thanassis Tsiodras
Julien Delange

December 17, 2012

Contents

1 Introduction

Taste currently stands for *"The ASSERT Set of Tools for Engineering"*.

In this tutorial you will learn how to build a system following the Taste philosophy. You will see how to make your *data view* and its associated *interface view* and *deployment view*. You'll also see how to generate the code and automatically create program that run on top of various operating systems such as Linux or RTEMS.

This tutorial is supposed to be used with the Taste virtual machine, that consists in a VMWARE© image that contains all the necessary tool. So, you must start first with the installation of the VMWARE PLAYER© software to use the Taste image. Then, you will have the complete environment required to run Taste tools.

2 Install with the VMWARE© image

Download the toolchain here: `http://www.semantix.gr/ASSERT-VM.tar.gz` It contains a full Linux installation with all Taste tools in the form of a VMWARE© image. You will need the VMWARE PLAYER© Player to open it, you can get it for free at `http://www.vmware.com/products/player/`.

The Linux installation of the VMWARE PLAYER© image contains a complete and up-to-date installation of Taste. Most of the tools are present in the `/opt` directory. A few others are in system directories.

Most new versions of the toolchain won't require a reinstallation of the VMWARE© image (which is very big).

The easiest way to work with the VMWARE© image is to connect to it using an SSH connection (`putty` on *Windows* - see 4.2, `ssh -X` on *Linux*). You can exchange files with the SCP protocol (e.g. WinSCP tool - see 4.2).

2.1 Login information

The user login of the virtual machine is

- **Login:** assert
- **Password:** assertvm

2.2 Updating the toolchain

When you need to update/replace tools in the `/opt` directory you need to use the `sudo` command. For instance, to replace an old version of the toolchain you need to follow these steps:

1. `cd /opt`
2. `sudo wget http://www.semantix.gr/assert/WP4.4-ToolchainAndManual-Linux-latest.tar.gz` (enter the sudo password)
3. `sudo rm -r WP4.4-ToolchainAndManual-Linux-yourversion`

4. `sudo tar zxvf WP4.4-ToolchainAndManual-Linux-latest.tar.gz`
5. `source ~/assert_env.sh`

The last command is very important. It will set your path and environment variables to the new toolchain.

2.3 Changing the keymap

You can set the keyboard mapping to your own using the following command (replace fr by your mapping of choice) :

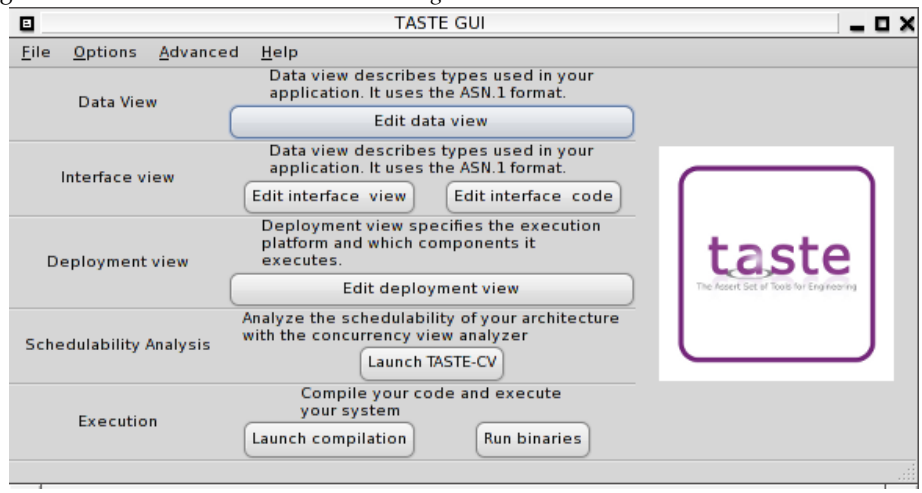
```
sudo setxkbmap fr
```

2.4 Stay tuned

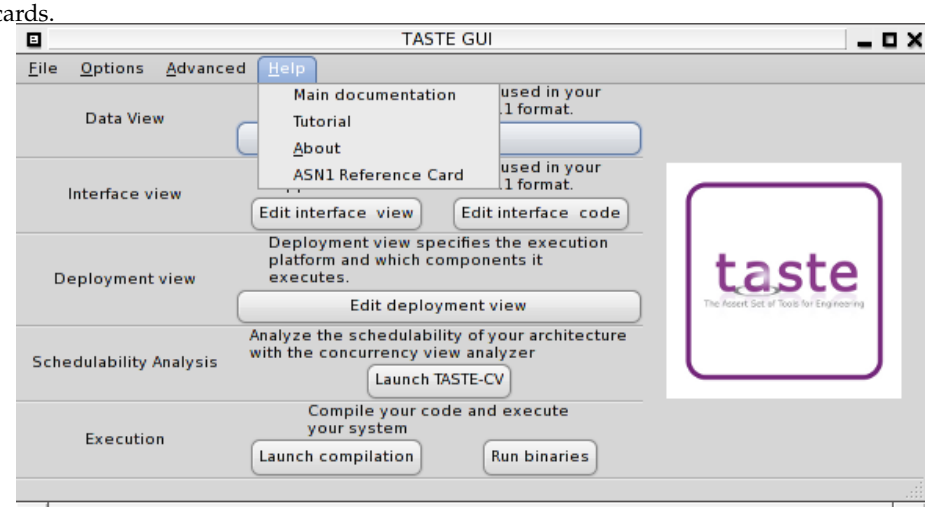
To get informed about releases of Taste, you can subscribe to the RSS feed on `http://www.semantix.gr/assert/assertToolchain.xml`

3 How to make a simple system

Inside the VMWARE PLAYER© image, you can find this tutorial but also a tool called *tastegui*. It is a graphical interface used to control all Taste components. This interface also contains all required documentation about Taste and its environment. The following figure shows the main window of the *tastegui* tool.



This tool is composed of several menus that will provide all the necessary documentation if you need help. In particular, the menu help (shown in the following figure) can be used to show the main Taste documentation as well as the ASN1 and AADL reference cards.



3.1 What the demo does

This section will guide you to build a simple system that runs on Linux. It is composed of two tasks : one task periodically sends an integer to another task. It is a *ping* example,

where one task periodically polls another one.

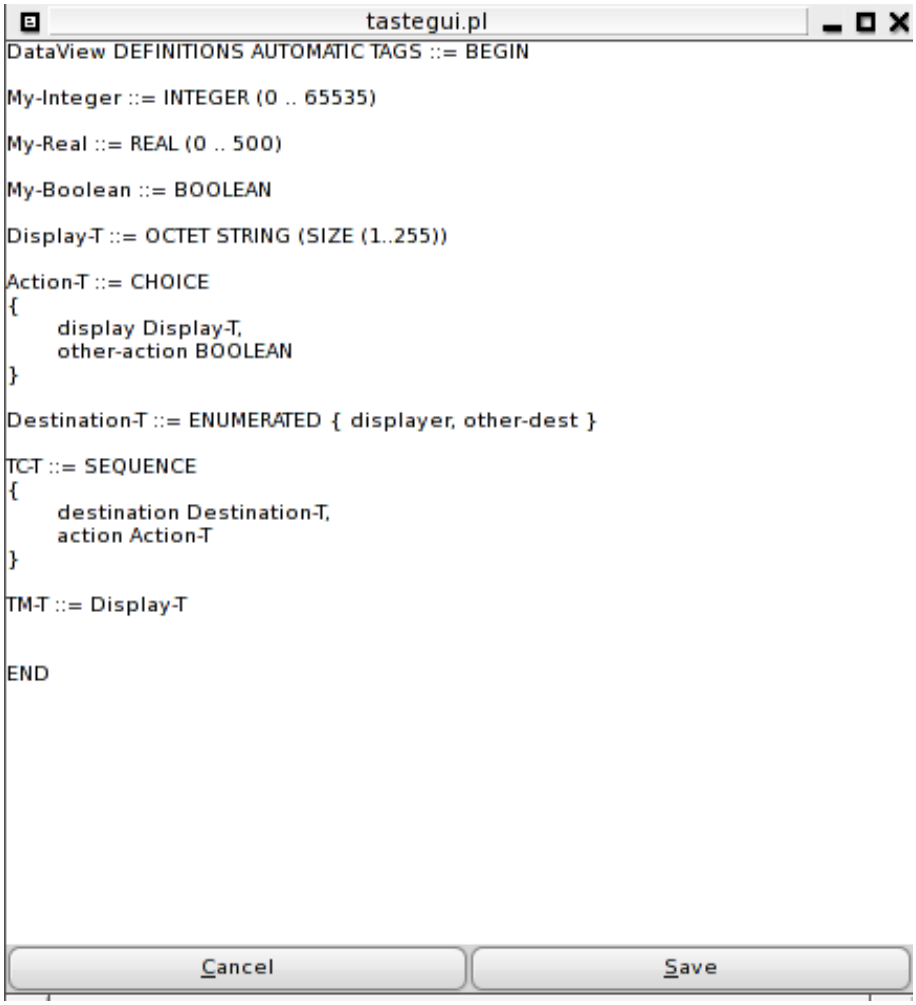
3.2 The process step by step

The Taste development process is composed on the following aspects:

1. **Define your data model**. The data model is the definition of all data types used in your system. In Taste, we use the ASN1 standard to describe data types. ASN1 is a standard widely used in the telecommunication industry and is very efficient to enable communications across heterogeneous systems (which have different data types representations).
2. Create your interfaceview. It describes your system from a pure functional point of view. It includes the definition of the functions of your system with its interface. Thus, it relies on the dataview to describe the data types used by each function of the system.
3. Write the code of the functions you defined in the interfaceview. Depending on the language used by each function, you will have to write it using a regular language (such as C or Ada) or provide reference to functional models (SDL, Simulink, etc.).
4. Design your deploymentview. It describes how functions are allocated in your architecture. You can bind functions of your interfaceview to different processors and connect them using software bus (such as ethernet).
5. Assemble and compile the system. In this step, the Taste toolchain auto-generate code that implements the architecture of the deploymentview and integrate your code on top of it. It integrates assemble each piece of code and produces the binaries for each system.

3.3 Creating the Data model in ASN.1

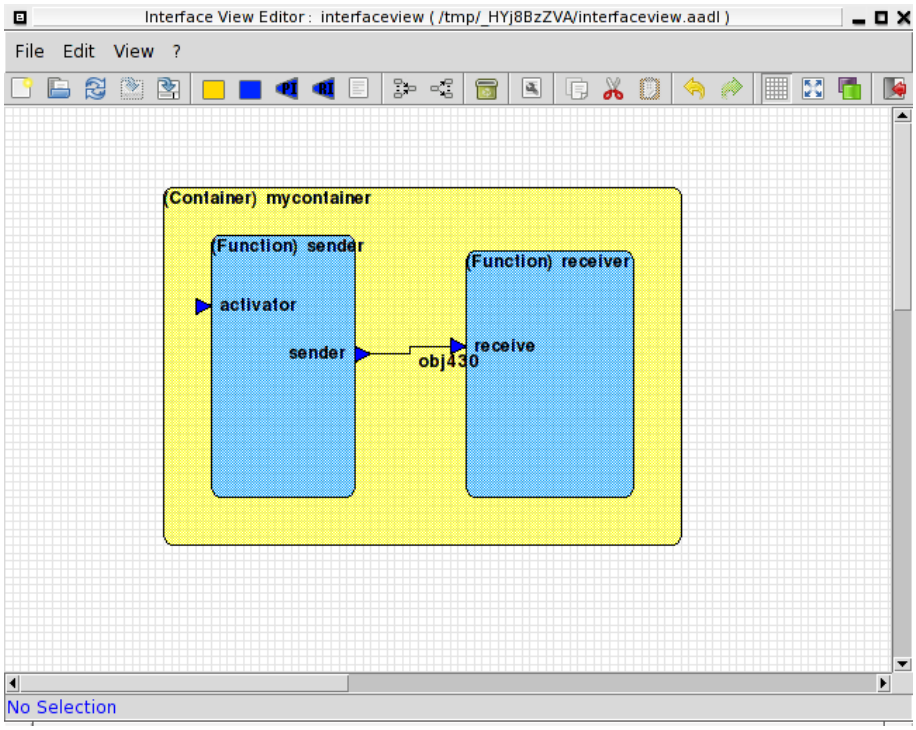
The standard ASN1 and the language is kept unmodified. You can write all the data types your need to express the parameters of your messages. To create your dataview, click on the button *"Edit data model"* in the *tastegui* tool. It will open a window to let you edit the definition of your data model. Once you have finished to edit it, save it. The *tastegui* tool will show you potential errors contained in your data model. The following figure shows the dataview editor.



To assist system designers in the use of Taste, the *tastegui* automatically defines a dataview with several predefined types. For this tutorial, you can keep the proposed dataview unchanged, the types already defined are sufficient to build the example of this tutorial.

3.4 Interface view

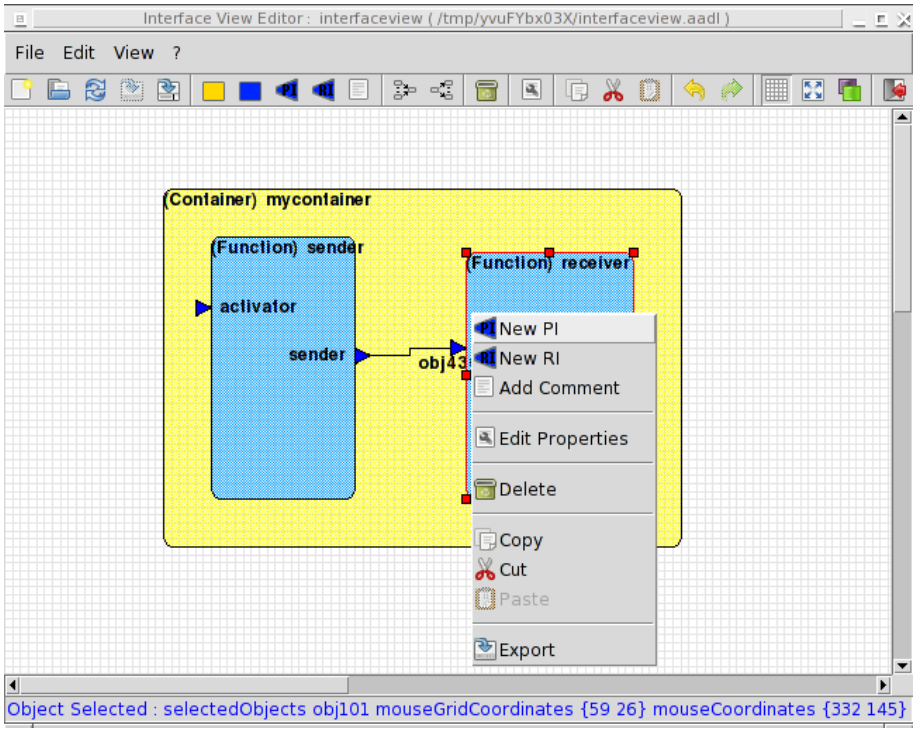
The interfaceview captures the functional aspects of your system. It lets you define the functions of your system with their timing characteristics (periodic, sporadic) and their interfaces (parameters that can be transmitted to the function). Taste includes a graphical editor to let you define your interfaceview. To edit your interfaceview, click on the button *"edit your interface view"* in *tastegui*. Then, a new window, as illustrated in the following figure should appear.



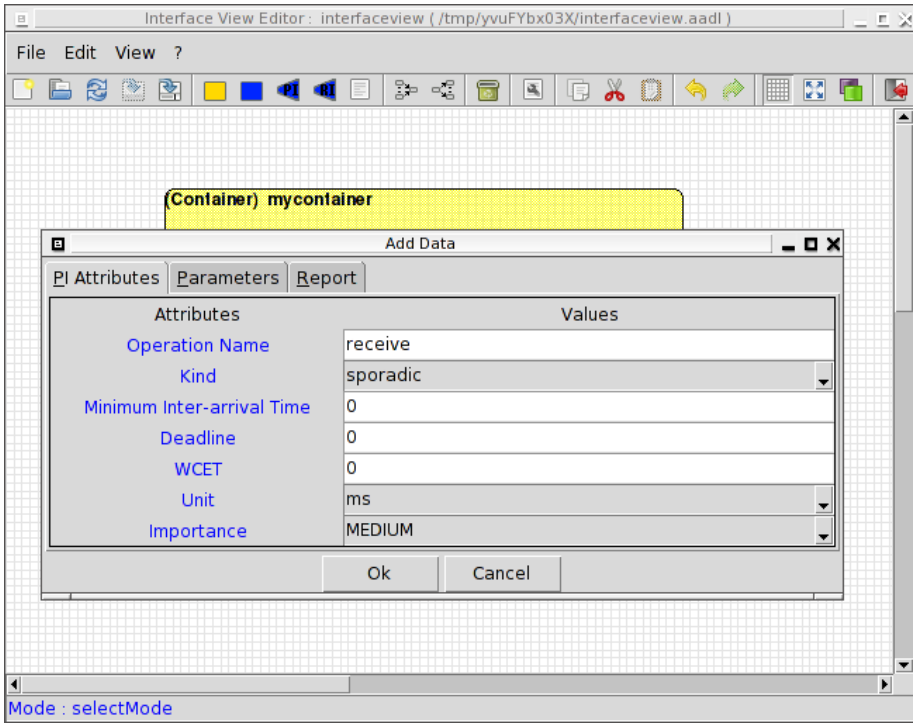
When you start the tool, the interface view is empty. First, create a container and call it `mycontainer`. It will contain the two functions that composed your system.

3.4.1 Receiver function

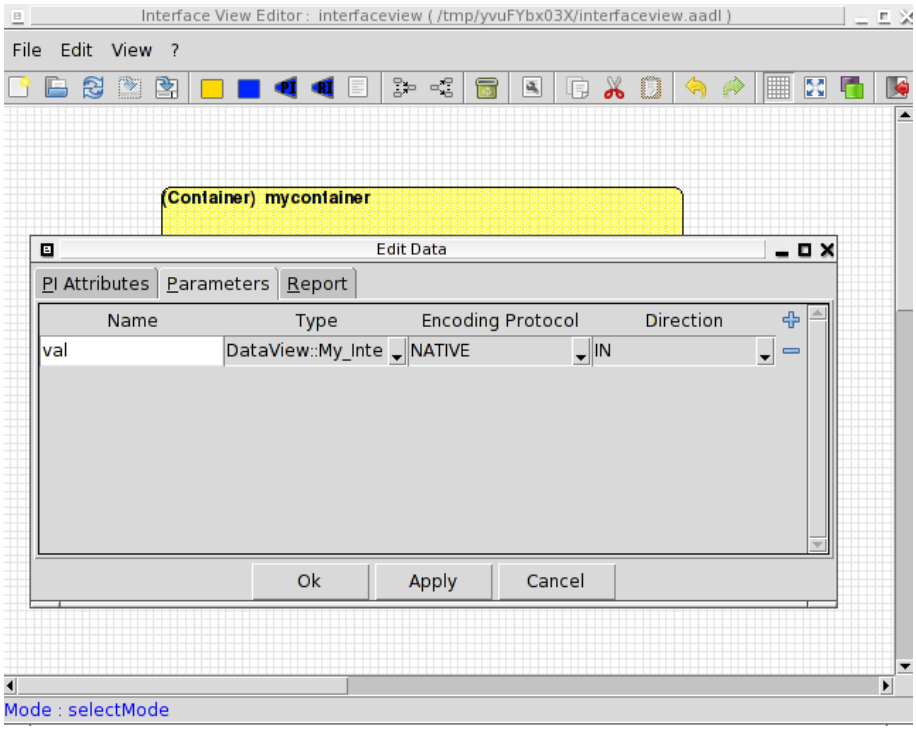
We will first define the function that receive data. Add a function in the container and call it `receiver`. Then, add a *Provided Interface* (called **PI**) to this function by making a right-click on the function, as depicted in the following figure.



Edit the **PI** properties. Define it as sporadic and define your own timing requirements. You should have a property window similar to the following figure.



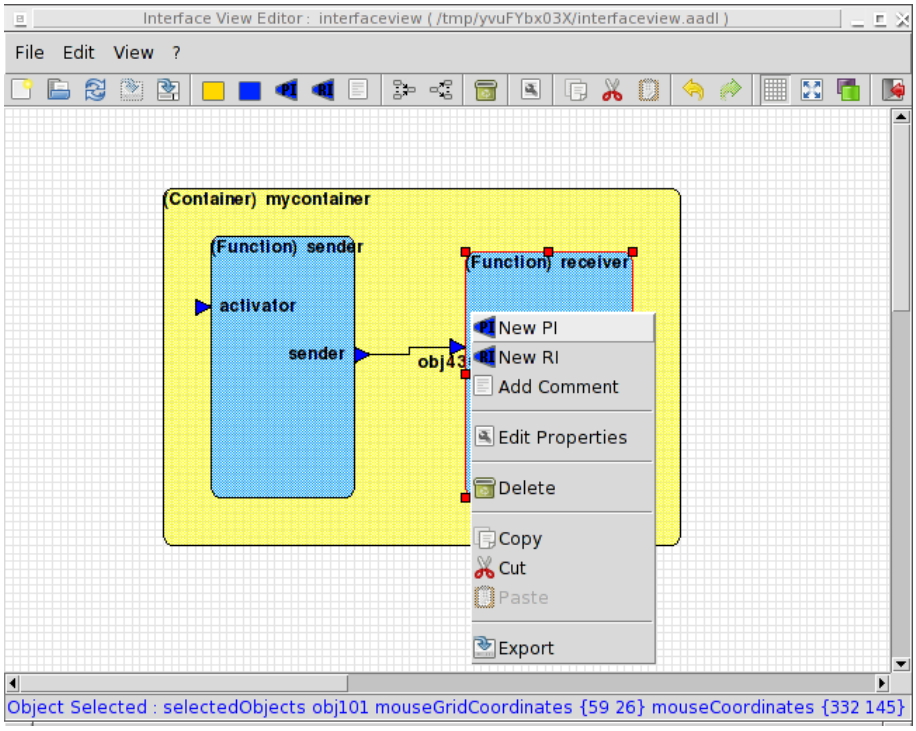
Then, you need to describe the parameters of your *Provided Interfaces*. You do that, edit the *Parameters* pane. Add a new parameter (by clicking on the plus sign). Set the name to `val` and associate it with the type `My_Integer`. Note that this type comes from the dataview. You should get a window similar to the following figure.



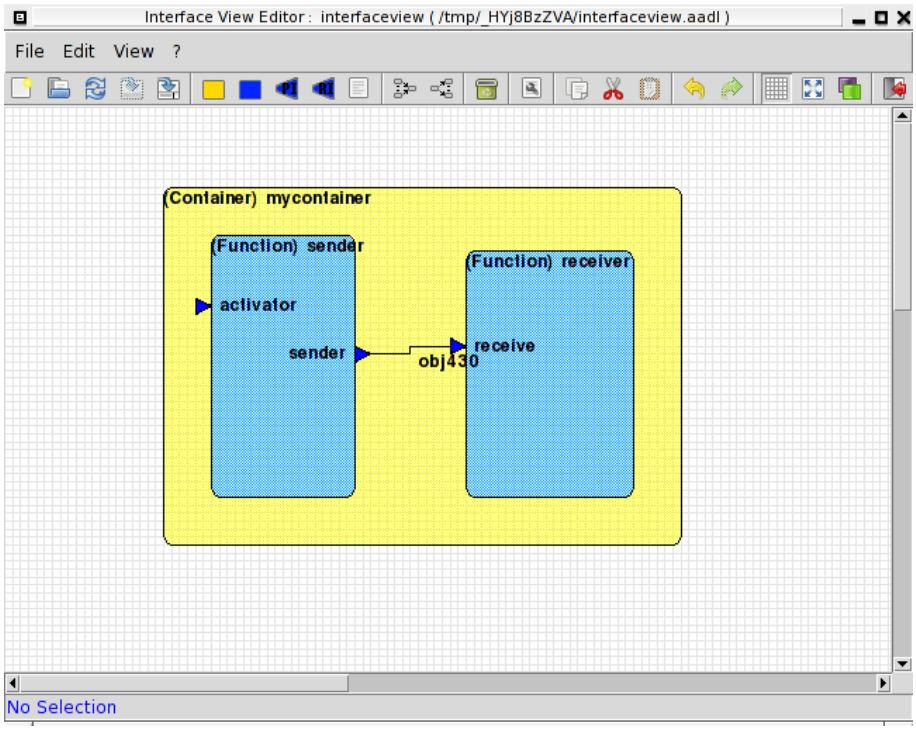
3.4.2 Sender function

Once the receiver function is defined, you need to define the sender function. Add another function in your container and call it sender. Then, add first a new *Provided Interface* (**PI**) and call it `activator`. This interface will be responsible to call the remote interface `receive` from the `receiver` function. In this **PI**, you define the period on which it is called as well as other timing requirements (deadline, etc.).

Finally, add a *Required Interface* (**RI**) to the `sender` function. Adding a *Required Interface* specifies that your function *needs* to be bind to another function. Its purpose is to associate interfaces, showing their dependencies and their interactions. Adding a *Required Interface* is similar to adding a *Provided Interface*: make a right click on the function and choose `Add RI`, as illustrated in the following figure.



Add the *Required Interface* in the `sender` function and call it `sender`. Finally, connect the `sender` interface from the `sender` function to the `receive` interface from the `receiver` function. You should finally get an interfaceview similar to the following picture.



Finally, save your interfaceview and quit the `TASTE-IV` tool.

3.5 Write the code of your defined functions

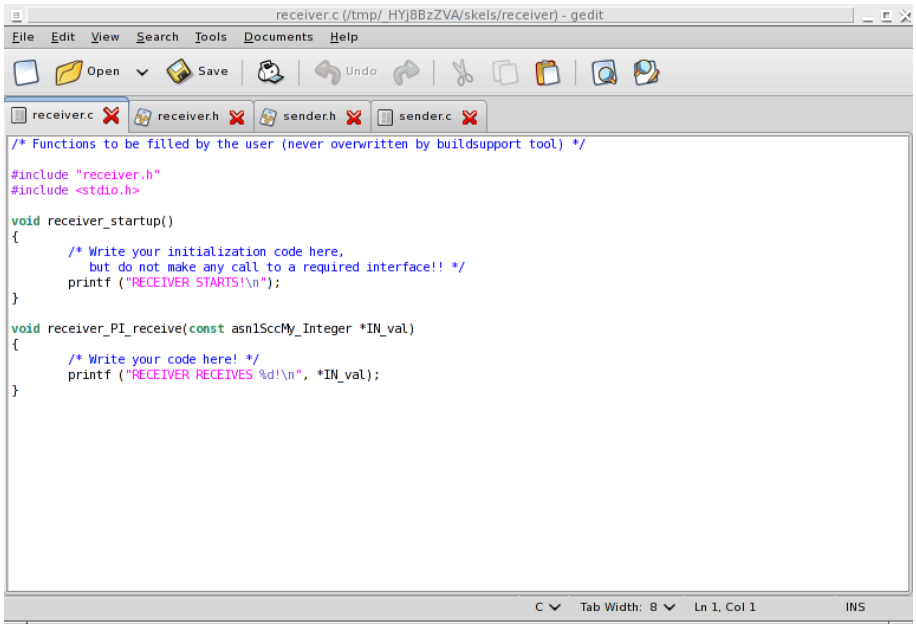
One you have defined system functions, you need to write the code. This code can be written using language such as Ada, C or application-level model such as SDL, Simulink or SCADE. In our example, we use the C language to write the behavior of the functions.

To edit the functions code, click on the button *"Edit interface code"*. It will open a new window with text files containing code skeletons. then, you need to edit them in order to fill the functions code.

3.5.1 Code of the receiver function

Edit the `receiver.c` file. The function `receiver_startup()` corresponds to the function called when the function starts. Then, the function `receiver_PI_receive()` corresponds to the function called when a data is received by the `receive()` interface. The parameter (`IN_val`) corresponds to the data received, encoded with the ASN1 standard.

Complete the code in order to have a similar code than the following figure. It will print a text line when the function starts and output the value received on the `receive` interface each time a data is received.

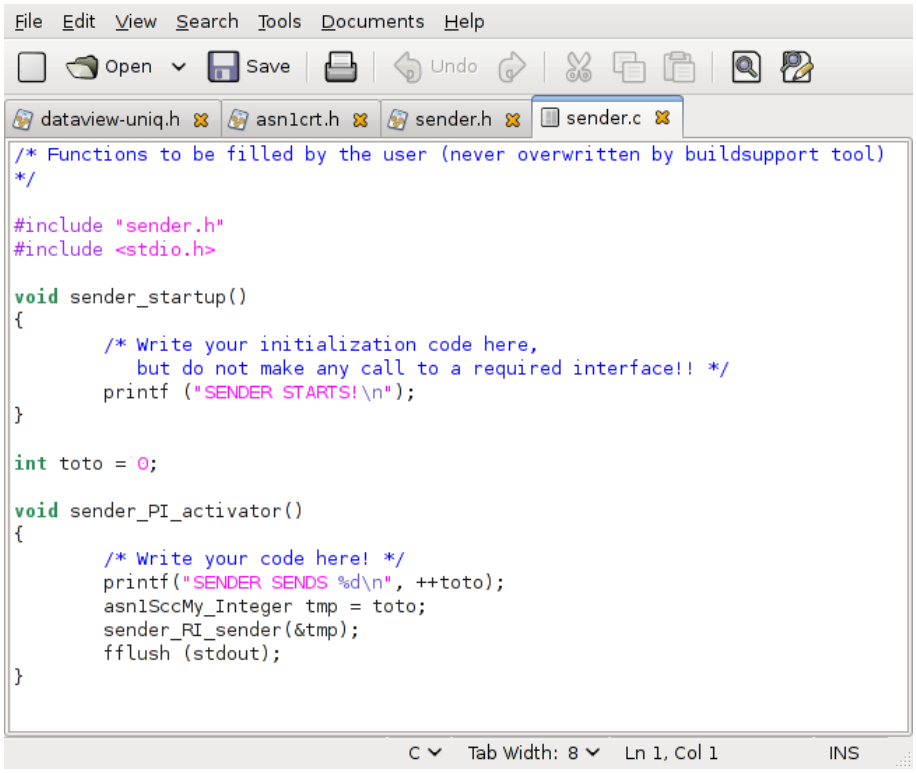


3.5.2 Code of the sender function

Edit the `sender.c` file. In this file, the function `sender_startup()` corresponds to the function called when the function starts. Then, the function `receiver_PI_activator()` corresponds to the function periodically called by the system. You have to complete this function by defining the code that will be called on a periodic basis by the system.

As you want to call the `receive()` interface of the `receiver` function, you have to use the *Required Interface* of your system. This required interface is called through a system call to the `sender_RI_sender()` function. When you call this function, it automatically calls the interface defined in the `receiver` function. So, add a call to this function and use a parameter that contain the data sent as parameter to the `receive()` function.

Complete the code in order to have a similar code than the following figure. It will print a text line when the function starts and will call the required interface periodically so that this sender function will automatically call the `receive` interface from the `receiver` function.

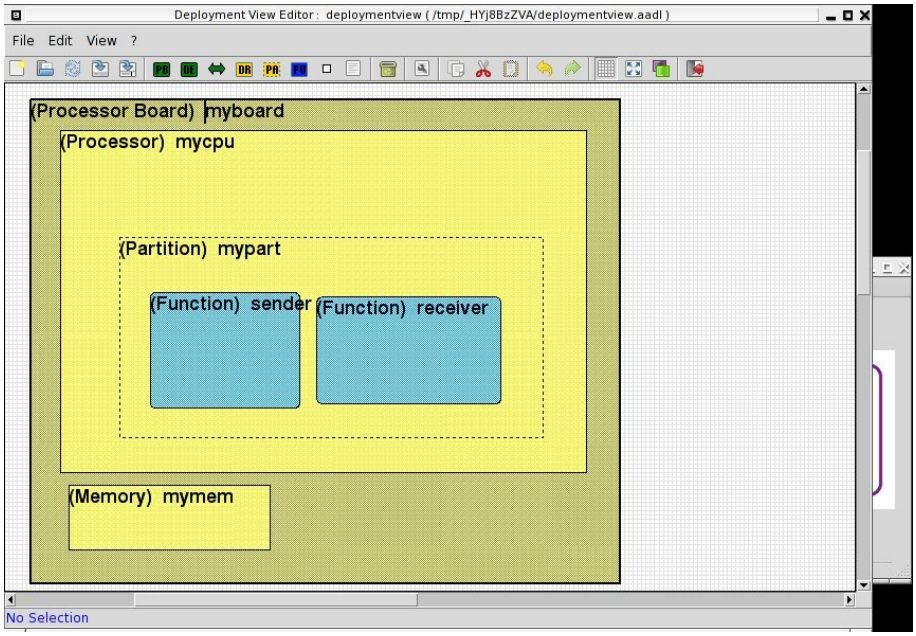


The function definition of all *Required Interfaces* can be found in the header file of the function. In our case, the definition of the function `sender_RI_sender()` can be found in the `sender.h` header file.

Finally, save your source files and quit the editor program.

3.6 Build the deployment view

This step consists in mapping functions to real hardware: you associate the functions of the interfaceview to processor and boards. To define your deploymentview, click on the button *"Edit deployment view"* in the main window of *tastegui*. It will launch a program that looks like the following picture.

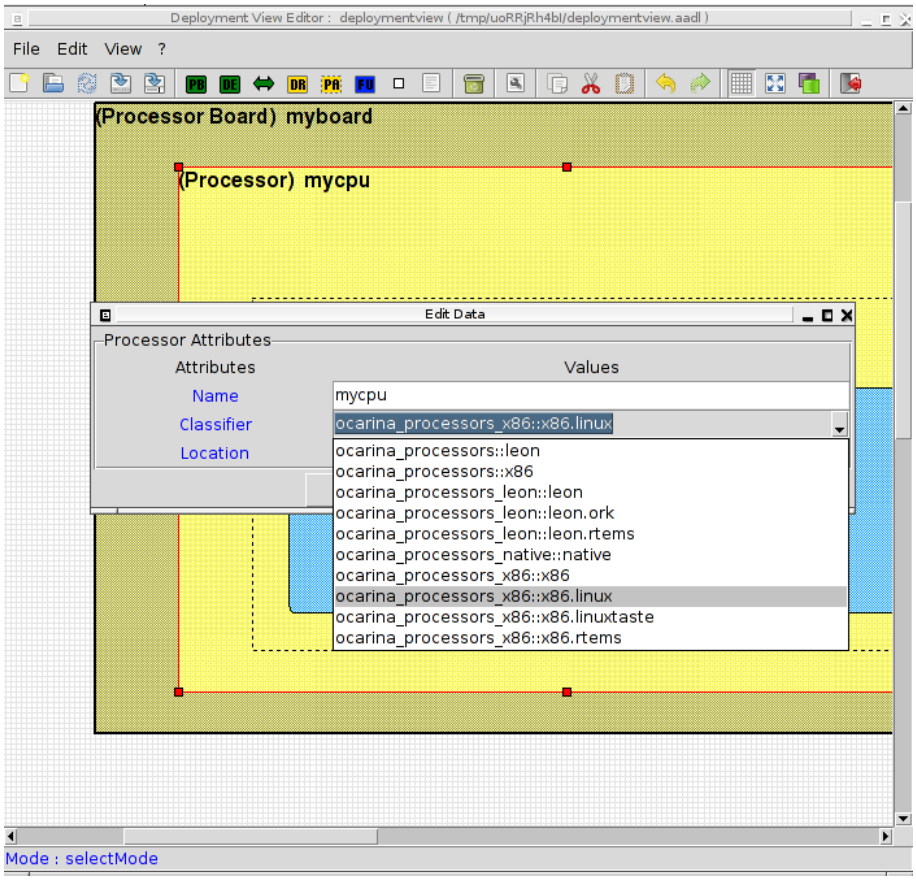


Then, add a processor board in your deploymentview and call it `myboard`. A processor board contains all hardware components required to execute a system: a CPU and a memory. Give a name to the processor (for example `mycpu`) and the memory (for example `mymem`).

The processor contains a partition. In the Taste approach, a partition contains functions from the interfaceview. A single partition can collocate several functions.

First, give a name to the partition defined in your processor (for example, `mypart`). Then, add the two functions of your interfaceview: `sender` and `receiver`. By doing that, you specify that the two functions you specify are executed within a single partition executed on the same processor.

Then, you need to define the properties of your processor: its architecture but also the operating system it uses to execute your functions. To do so, select the processor component and perform a right click on it. Then, edit its properties (*"edit properties"* option in the menu). The preferences toolbox proposes to define a classifier, that corresponds to the architecture used by the processor as well as its operating system. As we are executing this example on a regular *Linux* system, choose the option `ocarina_processors_x86::x86_linux` as shown in the following figure.

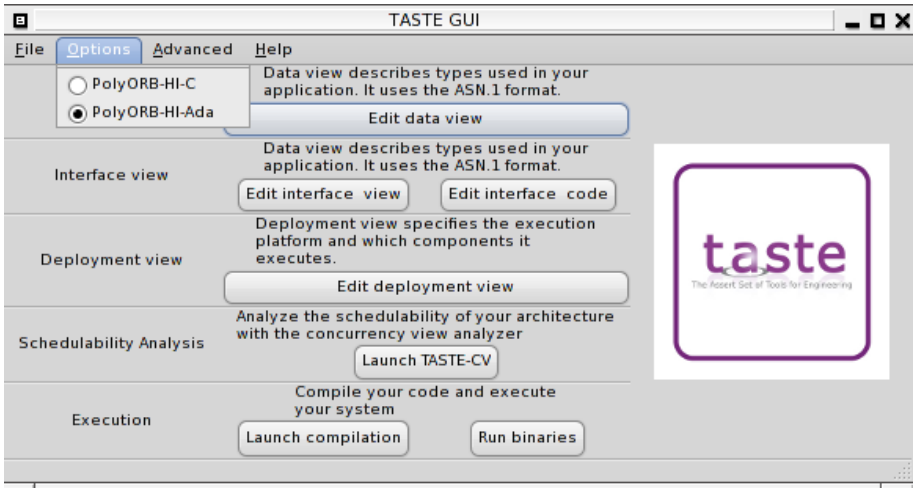


The resulting deploymentview should be similar to the one shown in the figure of the beginning of this section.

Finally, save your deploymentview and quit the `TASTE-DV` program.

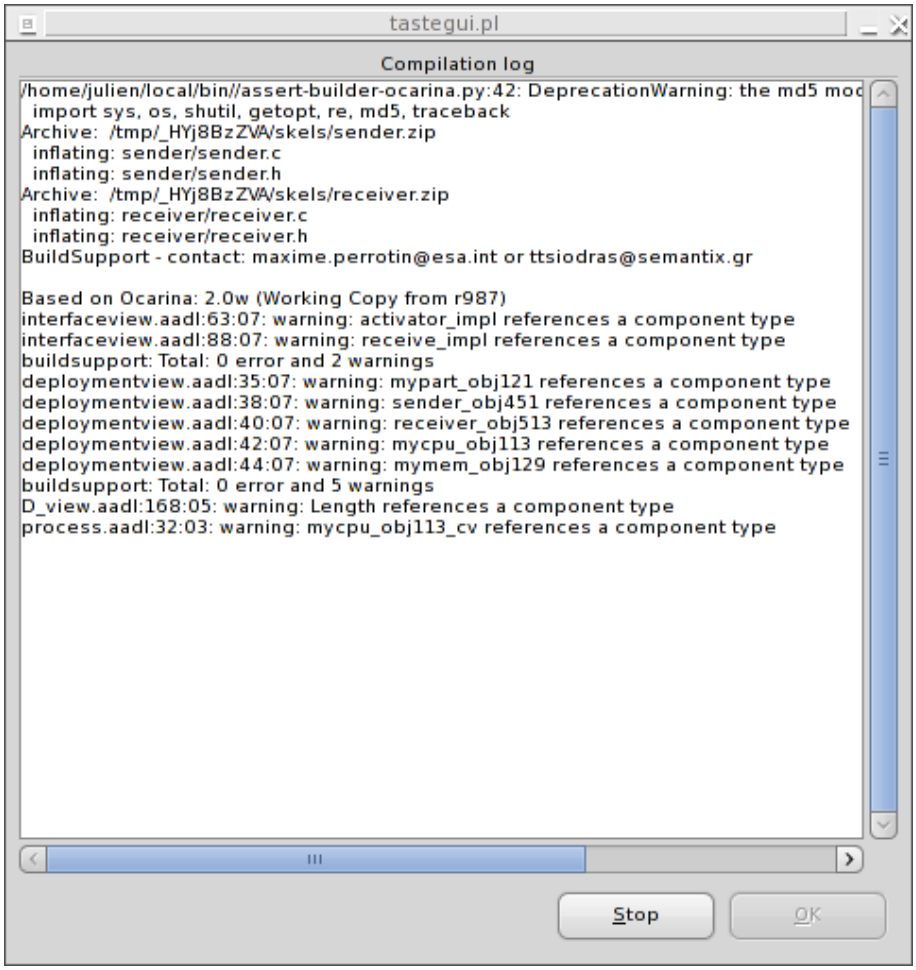
3.7 Build and execute the system

Once you have defined your interfaceview, your deploymentview and that the code of your functions is written, you are able to build your system. Before invoking the build functionnality, you have to define which runtime you'll use. Taste offers two runtime: PolyORB-HI-C and PolyORB-HI-Ada. You can choose it in the option menu, as shown in the following figure.

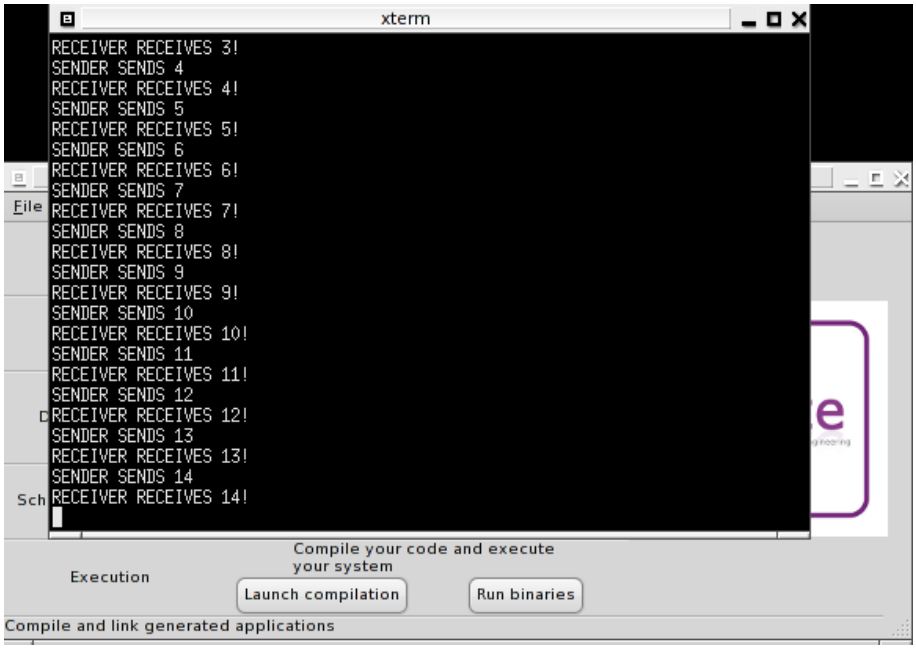


As our example use PolyORB-HI-C, **choose the option PolyORB-HI-C in the option menu**.

Then, click on the *"Launch compilation"* button. A window will appear, showing the output of the build process. During the compilation, the tool generate the architecture code, integrate your functional code and produce the binaries of your system. Note that if you have several processor boards, it will create the binaries for each of them. An example of the compilation window is shown in the following figure.

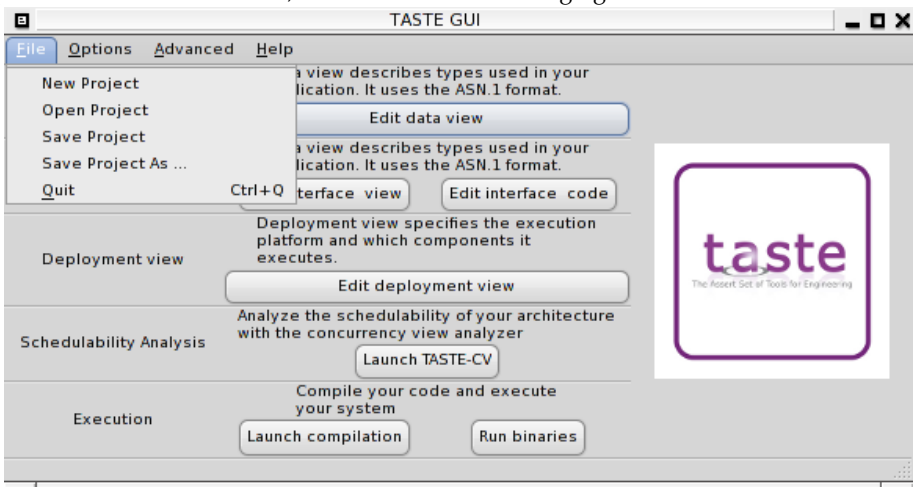


Once the compilation process is finished, you can run the produced binary by clicking on the button *"Run binaries"*. A window that execute your system will appear, as shown in the following figure.



3.8 Save your project

If you want to edit your project later, you can save it. To do so, choose the *"Save project"* option from the *"Files"* menu, as shown in the following figure.



3.9 Easily reproduce this tutorial

For people who want to reproduce this tutorial in a quick way, a complete existing project is available in the archive. Just open the `project.taste` file using the *"File"* menu : it

contains a complete dataview, interfaceview, deploymentview as well as functional code so that you are able to build and execute binary without writing anything.

3.10 Get more information & support

This tutorial gives a simple overview of basic functions of the Taste toolchain. However, it is composed of more functionnality: it can integrate Ada code, Simulink/SCADE/SDL models, execute systems on top of various embedded operating such as RTEMS or also run functions on heterogeneous processors (such as LEON, PowerPC and so on). You can find more information about the toolchain in the main documentation, available from the *"Help"* menu. It is also accessible on the SEMANTIX website (see section).

If you experience problems during the use of Taste and its toolset, please send a mail to the development team with your project file. They will help you in your use of the Taste toolchain.

3.11 Additional feature: schedulability analysis

You can automatically perform a schedulability analysis test of your system by using the `TASTE-CV` tool. To do so, click on the *"Launch TEST-CV"*. The *tastegui* tool will load the definition of your system (with its tasks, scheduling and timing constraints and requirements) so that you are able to perform a schedulability analysis. However, this aspect of the toolchain is not covered by this tutorial, you can get more information about it in the main Taste documentation.

4 Resources

4.1 More information

- The ASSERT project: `http://www.assert-project.net`
- SEMANTIX website: `http://www.semantix.gr/assert`

4.2 Useful programs

- GNAT compiler : `http://libre.adacore.com`
- Gnatforleon : `http://polaris.dit.upm.es/~ork/`
- PuTTY : `http://putty.very.rulez.org/download.html`
- RTEMS: `http://www.rtems.com`
- SWIG : `http://www.swig.org/`
- WinSCP : `http://winscp.net`
- WxWidgets : `http://www.wxwidgets.org/`