



Getting Started User Guide

This document is intended as a “getting started” guide to eSSYN. As such, reader will find a mix of eSSYN GUI usage instructions and basic UML-MARTE modelling concepts. In order to keep a manageable document complexity all modelling capabilities are not described in detail and frequent referencias to the Modelling Methodology Reference document [1] are made for further consultation on specific details.

Introduction

eSSYN is a software synthesis tool for embedded systems. With eSSYN, and starting from a standard based application and platform model, embedded system designers can generate a complete set of target binaries for an embedded application in a matter of minutes. The tool will significantly shorten development time and allow for future reuse.

In order to use eSSYN, system designers need to provide:

- A software component based model of the application. As part of this model designers need to provide functional code (C/C++) for the components.
- A model for the hardware platform that specifies the available resources (mainly number and type of cores and operating systems).
- Mapping of software components and cores.

eSSYN will generate:

- All required code and system calls implementing communications among software components.
- All required makefiles for compilation.
- After compilation (from eSSYN environment) as many executable files as specified for the required cores and OS, ready to upload to the HW platform.

eSSYN provides a model generation wizard that makes first time model creation for a system painless, avoiding the need to learn the underlying modelling standard UML/MARTE.

eSSYN has been developed as a plug-in for Eclipse+Papyrus. The GUI you will find will be the standard ECLIPSE with Papyrus menu and a dedicated eSSYN tab (shown as PHARAON).

Basic modelling concepts

The system model is made of three sub-models, following the 'Y' structure (Figure 1) common in current design methodologies. The two branches of the 'Y' are the separate models for SW application (PIM) on one side and platform (PSM) on the other. Both models are connected by the PSM that defines the mapping of SW into HW.

A Model Centric Methodology uses a System Model as the center repository for the description of the system. All the actions performed as part of the design flow uses a unique model that is being enriched as the design flow progresses. Using such a reference model for specification, verification, performance estimation and synthesis is the way to go on embedded systems development due to the productivity increase and reuse potential of the methodology.

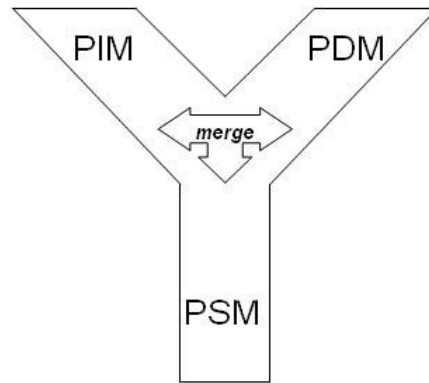


Figure 1 Y structure

The first task to perform will be to generate the three sub-models for our application and our platform:

- The Platform Independent Model (PIM), describes the functional and non-functional aspects of the application (e.g. application components, application structure, interfaces...). This models the application independently of the target platform. Therefore we can reuse it when implementing the application on other platforms
- The Platform Description Model (PDM), describes the different HW and SW resources that compose the HW platform. They are application independent elements. Of course a PDM for a certain platform can be reused for different applications or imported from other users.
- The Platform Specific Model (PSM), which describes the system architecture and the allocation of platform resources. This model allocates system functionalities to the platform resources in which they are implemented.

The three sub-models are built as a collection of “model elements” each. Examples of model elements are a data type, a sw component, a connector between components, among many others. Of course there will be a number of data type elements, each describing one data type used in the system, a number of component elements, and so on.

Model elements are organized in views. Different kinds of elements model different aspects of the system. In order to keep the model well organized, the elements are grouped in *views*. Each view deals with a specific aspect of the model description. In Figure 2 we can find the views related to each of the sub-models. A description of each view can be found below.

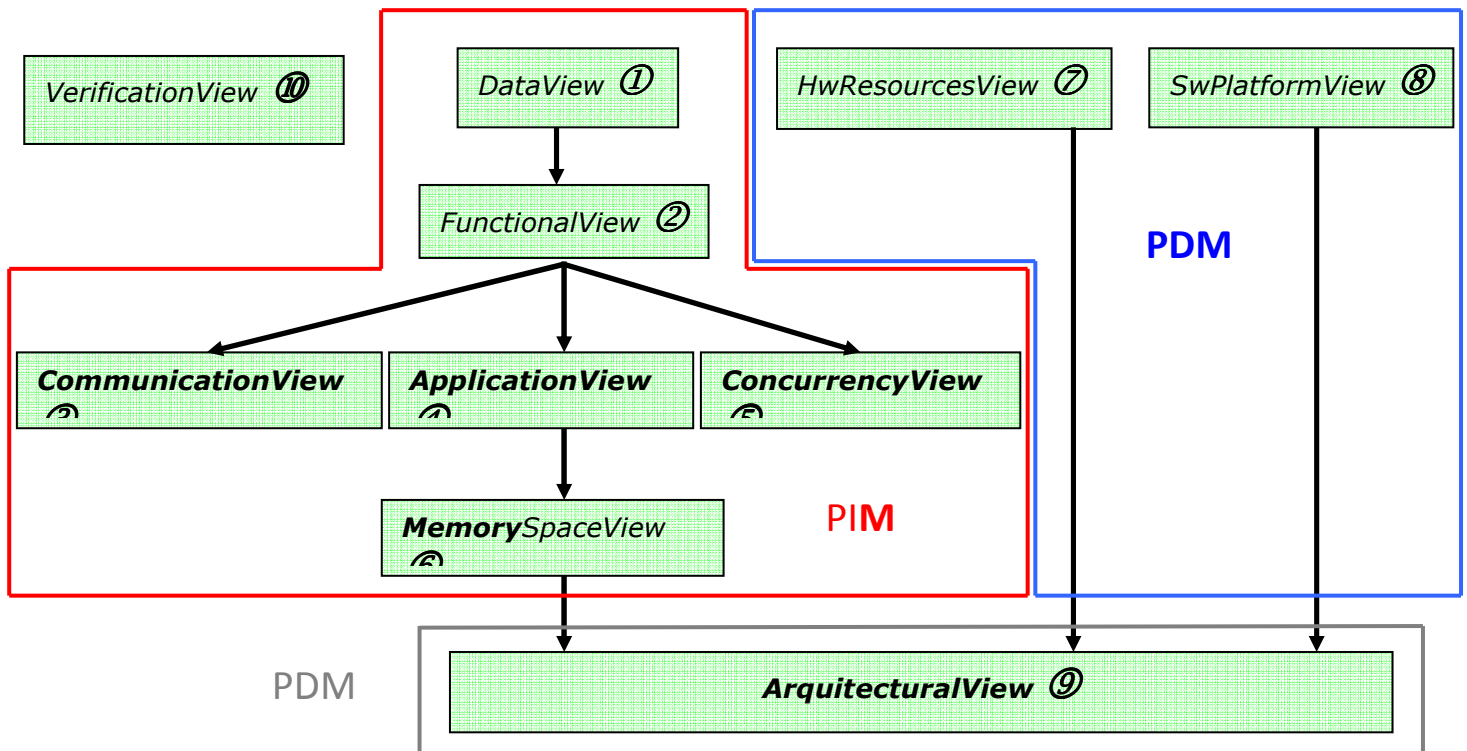


Figure 2 UML/MARTE view modelling activity

PIM Views

1. **Data View:** Defines the data types used by the information exchange among the system functionality.
2. **Functional view:** Includes the specification of the interfaces provided/required by the application components in order to be connected among them. It also specifies the files that contain the implementation (functional source code) of each application component.
3. **Application View:** Includes the definition of the application components and the application structure. Additionally, the view includes the association of the functional files defined in the *FunctionalView* to each application component. The view contains a “System” component which is used for specifying the application structure. It includes application components interconnect using the interfaces defined in the *FunctionalView* and the communication mechanisms defined in the *CommunicationView*.
4. **Concurrency view:** this view includes all the threads of the system. Additionally, this view includes the association of the application components onto these threads. The view contains a “System” component which is used for specifying the threads presented in the model and the mapping of the application components on these threads.
5. **Communication view:** captures the set of communication channels used for interconnect the different application components. Additionally, the view includes the mechanisms used for synchronizing threads and processes. The view is optional if no communication media are considered.

6. **Memory Space view:** defines the memory partitions that model the system processes as well as the allocation of application components onto these processes.

PDM Views

7. **HW Resources view:** provides a description of the processor cores available in the platform, the buses and other HW resources.
8. **SW Platform view:** provides a description of the SW platform resources (Operating Systems)

PSM Views

9. **Architectural view:** defines the platform architecture and the mapping of system processes onto platform resources. Additionally, this view includes the association of threads to processors. Mandatory to be included in the model.

External to the platform

10. **Verification view:** defines the environment components that interact with the system. Not mandatory.

Stereotypes

A Stereotype is a container of properties (e.g. attributes) that when applied to a model element formally describes the behavior of the element. Depending of the element, the applied stereotype will contain different properties. For instance, the stereotype applied to a Data Type will contain different attributes than the one applied to a communication channel. Stereotypes are NOT created by the user. The user chooses the right stereotype for the element from the available ones

A detailed list of attributes and properties included in each stereotype from the lists above can be found in [1].

Building a model

In order to build a model basically all the required model elements need to be described, grouped in the views defined above. The definition of the behaviour of each element is done by assigning the right stereotypes (one or more) to the element and defining each attribute in the stereotype.

Flow Overview

eSSYN design flow is very simple. The most complex stage is the model creation, and most of this document is dedicated to that.

1. Open ECLIPSE-eSSYN .

2. Project setup

Create UML project and assign Standard, MARTE and PHARAON profiles

2. Model creation: PIM, PDM and PSM

Create views, one by one and assign the right view stereotype

For each view, create the right elements

Assign the right stereotype to each created element

Assign values to the properties/attributes within the stereotypes

When required, create relationships between elements

3. Synthesis

Project Setup

Creation of an UML project

For creating a Papyrus project, click with the right button on the “Project Explorer” window and select “New” and “Project”.

Select “Papyrus” folder and, after that, “Papyrus project” (Figure 3).

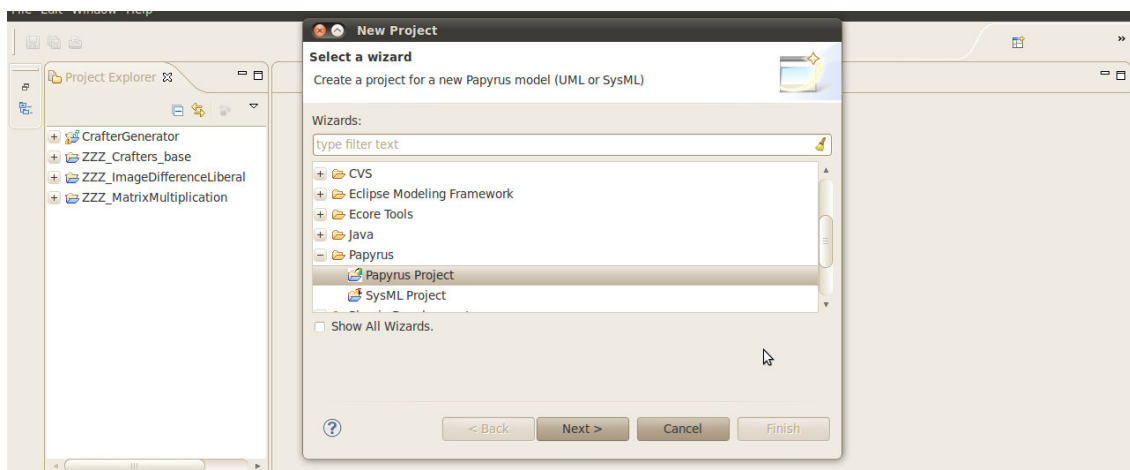


Figure 3 New project menu

Click “Next”, provide a name for the project and select the UML option for “Diagram Language”. After this, press Finish (Figure 4).

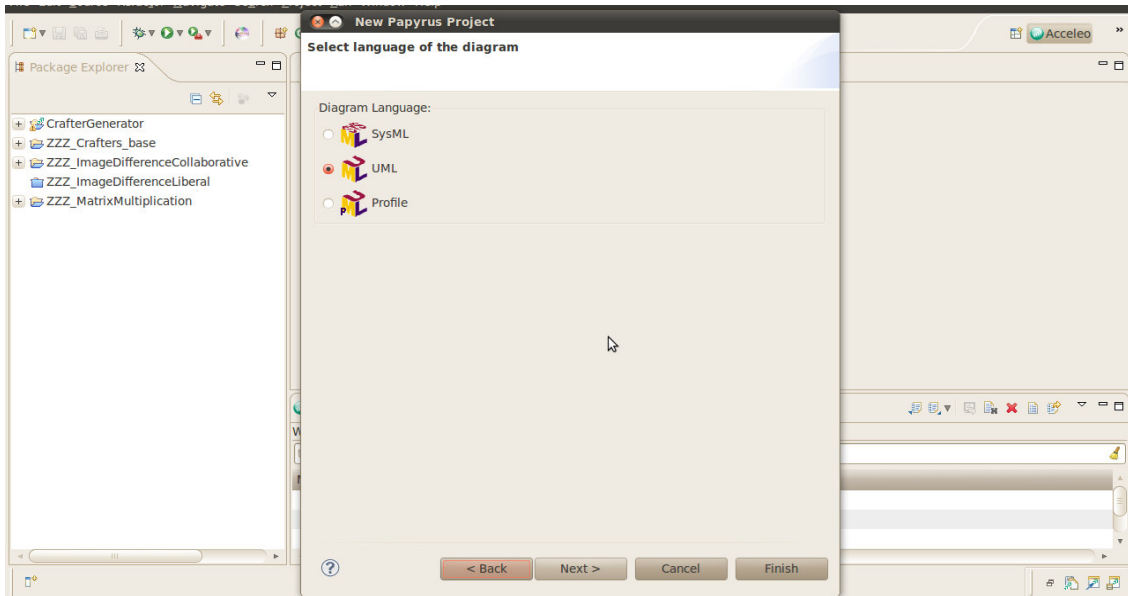


Figure 4 Characterize the Papyrus project

On “Package Explorer”, open the folder of the project. Double click on the “model.di” file (Figure 5). A diagram will open.

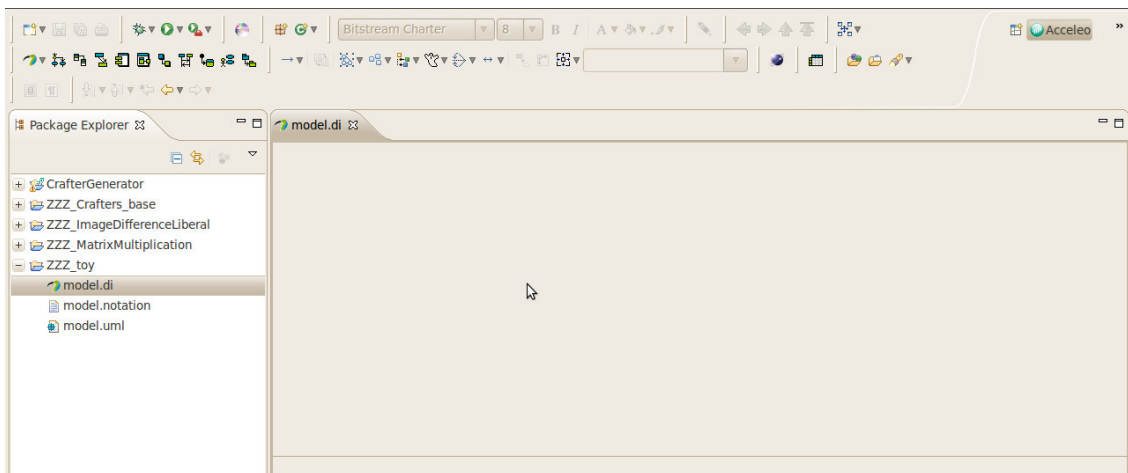


Figure 5 Open the model

Open the Papyrus perspective; on the right top, click on “Open perspective” (Figure 6).

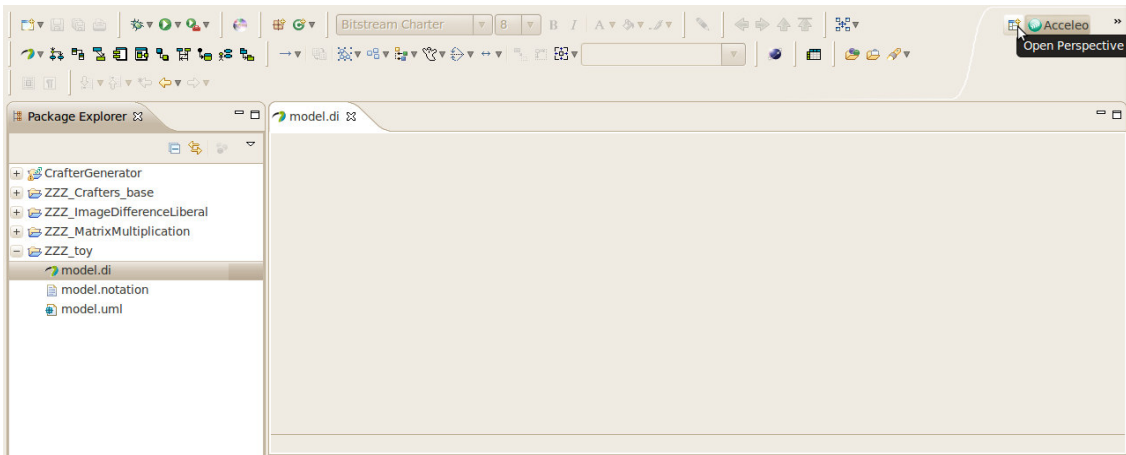


Figure 6 Change the Eclipse perspective to “Papyrus”

Select the Papyrus perspective (Figure 7).

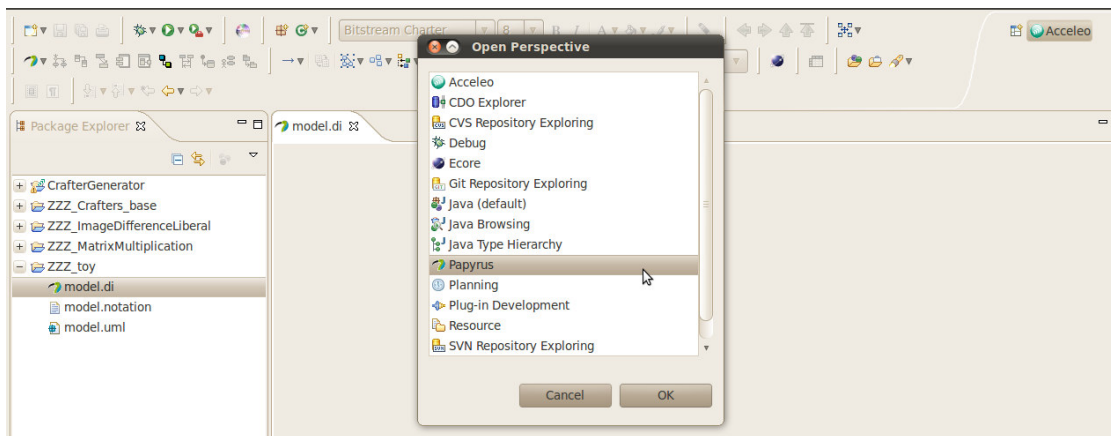


Figure 7 Select the Papyrus perspective

Figure 8 shows the result of applying the Papyrus perspective.

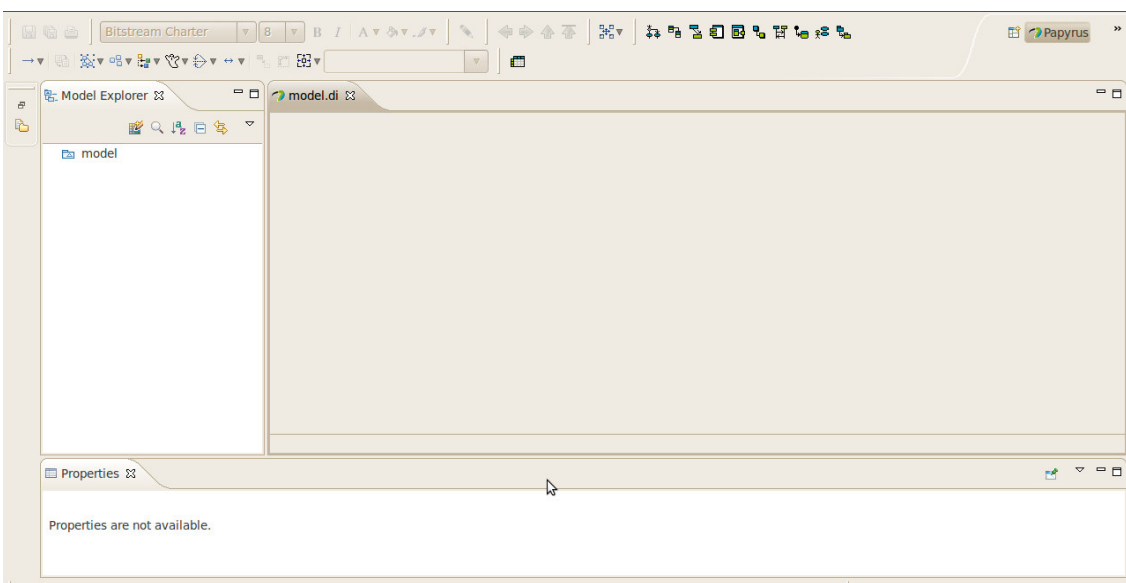


Figure 8 Papyrus model

The “Properties” window enables to see the properties of a model element: name, type, stereotypes applied... (bottom window of Figure 8). Go to Window → Show View → Properties

The “Model Explorer” window enables to navigate through the model (LEFT on Figure 8). In order to open Model Explorer, go to Window → Show View → Model Explorer

Load of a UML profile in a Papyrus UML model

Load a profile installed as Eclipse plug-in:

- In Model Explorer window, select “model”.
- In the Properties Window, select the tab “Profile” and click the shown inside

the red circle (Figure 9).

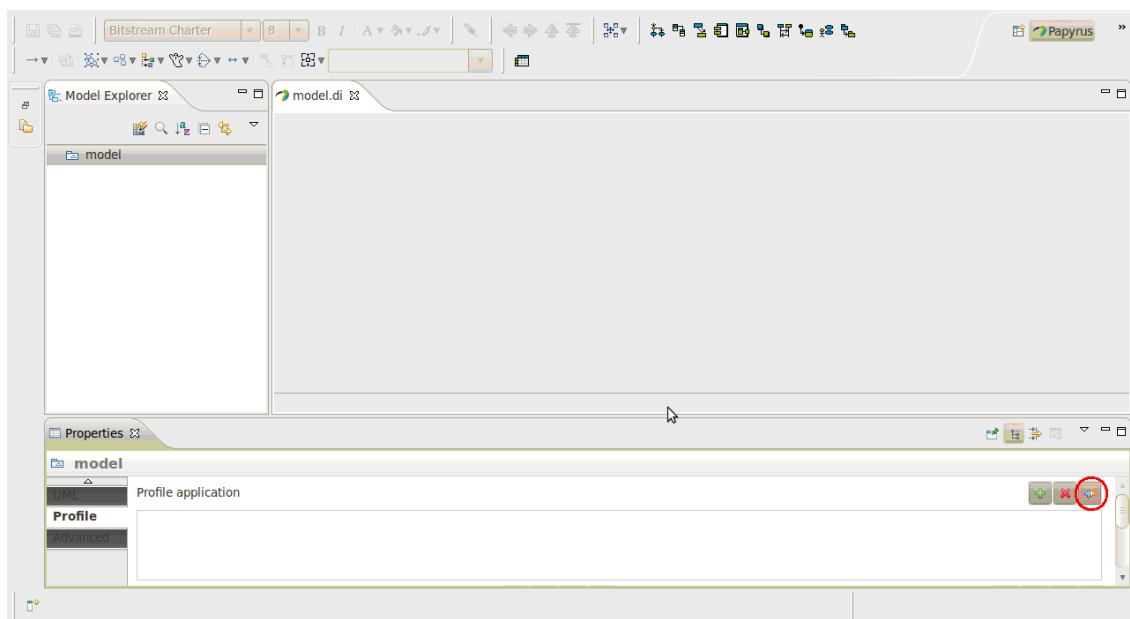


Figure 9 Load an UML profile

- Appears the next window where we select the profile to be used in the model (Figure 10). The profiles that are required to load are Standard, MARTE and PHARAON, one after the other. The box next to the profile name needs to be ticked.

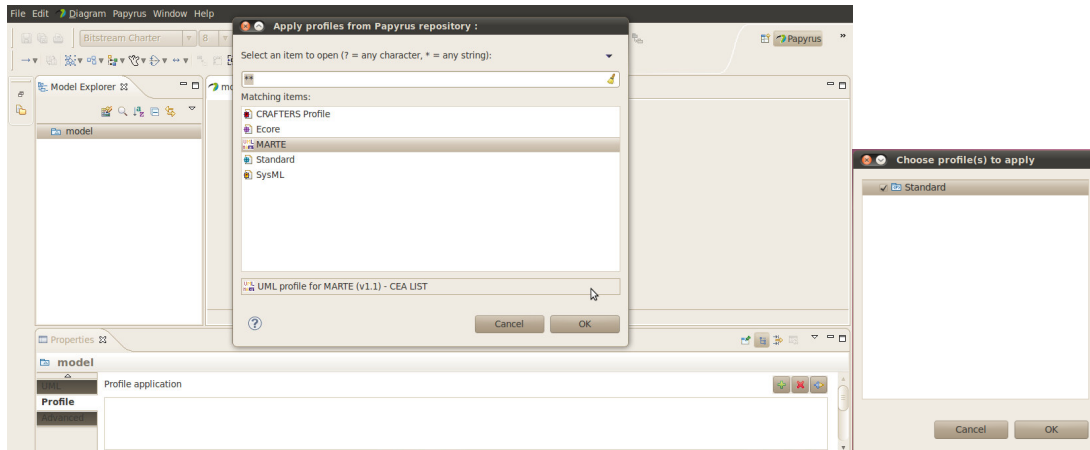


Figure 10 Load an UML profile

Now the project and setup is ready to start populating the model with the views and elements.

The Papyrus environment is based on “Diagrams”. All elements are created as graphical items in a diagram.

Model Creation

Basic actions

The following actions (create, stereotype) are done in the same way for all the views and elements within the views.

Creating a view

For creating a model view, click right button on the model element (Figure 11); go to “New Child” and select “Create a new Package” (Figure 11).

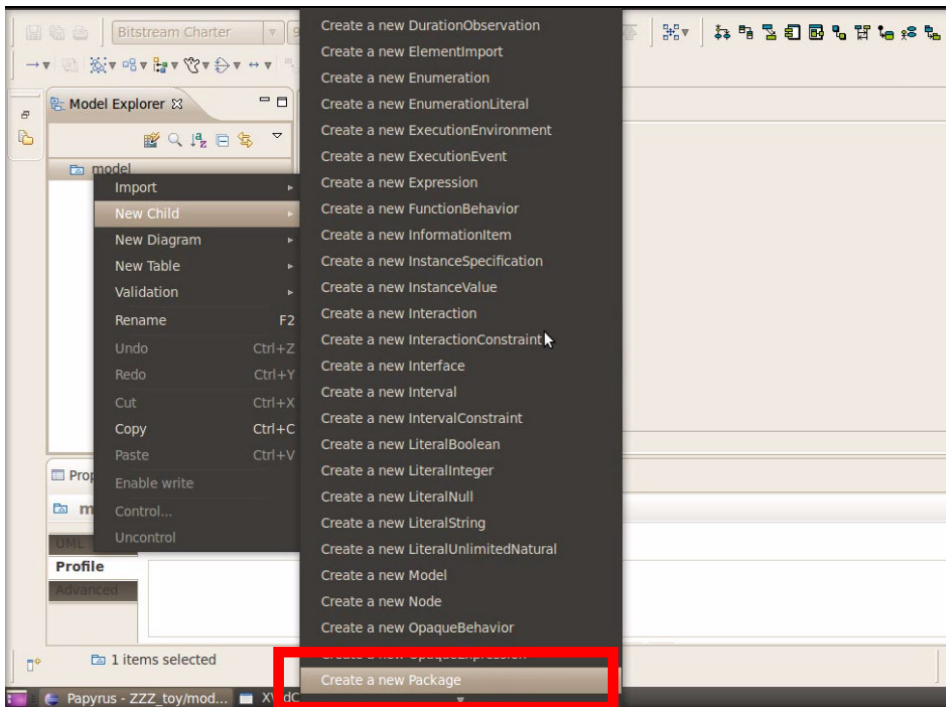


Figure 11 Create a UML package

Then, rename it from the default name “package” to a suitable name (right click on the name → rename) and stereotype it:

Click over the mode element and go the Properties → “Profile” tab (Figure 12). Then, on the right toolbar (Figure 12) go down until appears the tab “Applied Stereotypes”. Click on green crossed button (Figure 12). After that, a window appears (Figure 12). In this window, select the stereotype to be applied and click “ok” button.

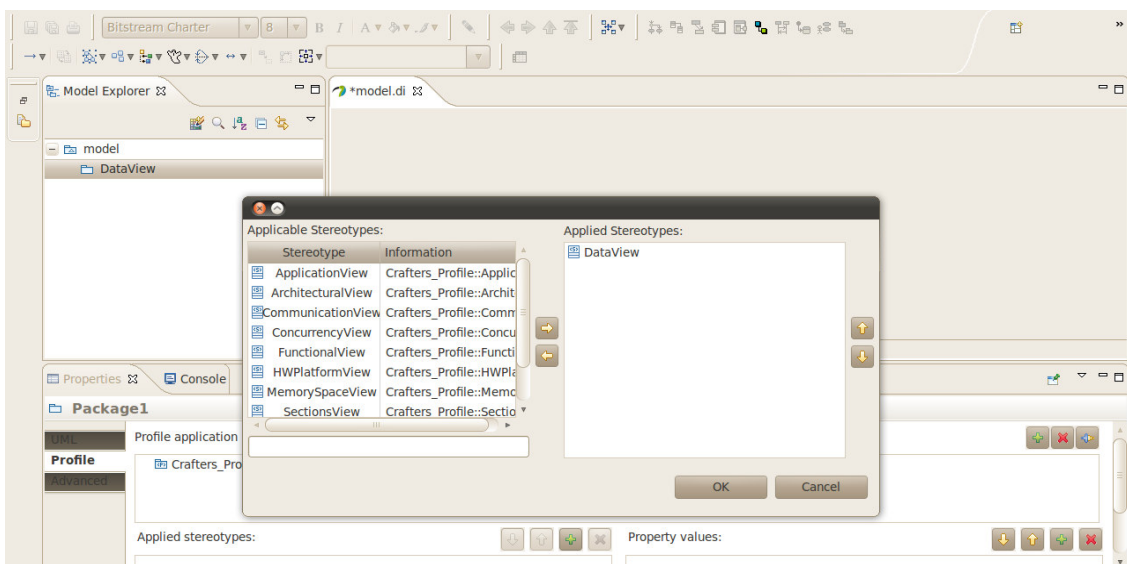


Figure 12 Applying a stereotype

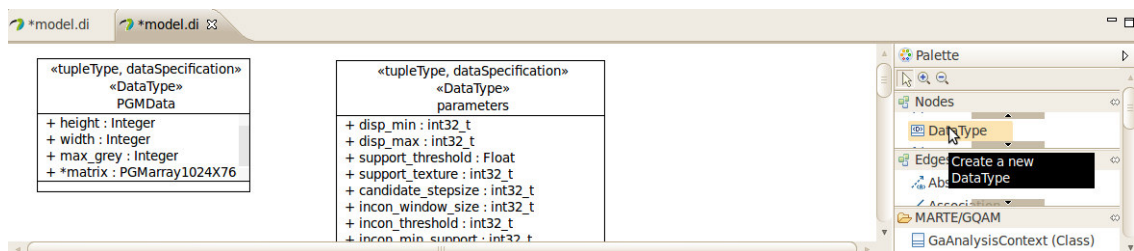
This has to be done for each of the views used in the sub-models (Data View, Functional View, etc).

Creating elements in the view

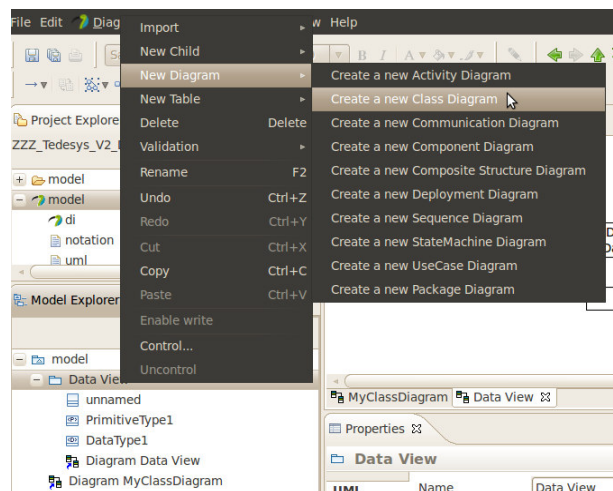
Once our required views are created and stereotyped, elements can be added into the views. Elements are placed in diagrams. Diagrams have not semantic meaning, and are just organizational. Diagrams have to be created below the view to which the elements belong to. Under a view as many diagrams as required for clarity can be created. In the Model Explorer window, right click on the desired view, then New Diagram → Create a new Class Diagram and give it a meaningful name (referred to its contents, e.g “structures”, “enumeration” and so on for Data View.

In order to add elements to a view, they are placed into one of the diagrams under the view. This can be done in two ways:

1. From the palette attached to the diagram, find the desired element, click, drag and drop it on the diagram.




2. From the Model Explorer, right click on the View, New Child → Create a new <your_element>. (Mouse control of the list is faulty. Better use arrow keys to navigate the list). The new element appears under the view. Click and drag to place it into the diagram.



Creating elements from the palette is more convenient, but not all element types are available there.

Once your new element is created:

- Assign a name: Select the “UML” tab in the “properties view” and type the new name.
- Stereotype it: Select the element from the diagram. Select the “Profile” tab in the “properties view”. Scroll to the bottom until you see the “Applied Stereotypes” section, click  and choose the right stereotype from the list.

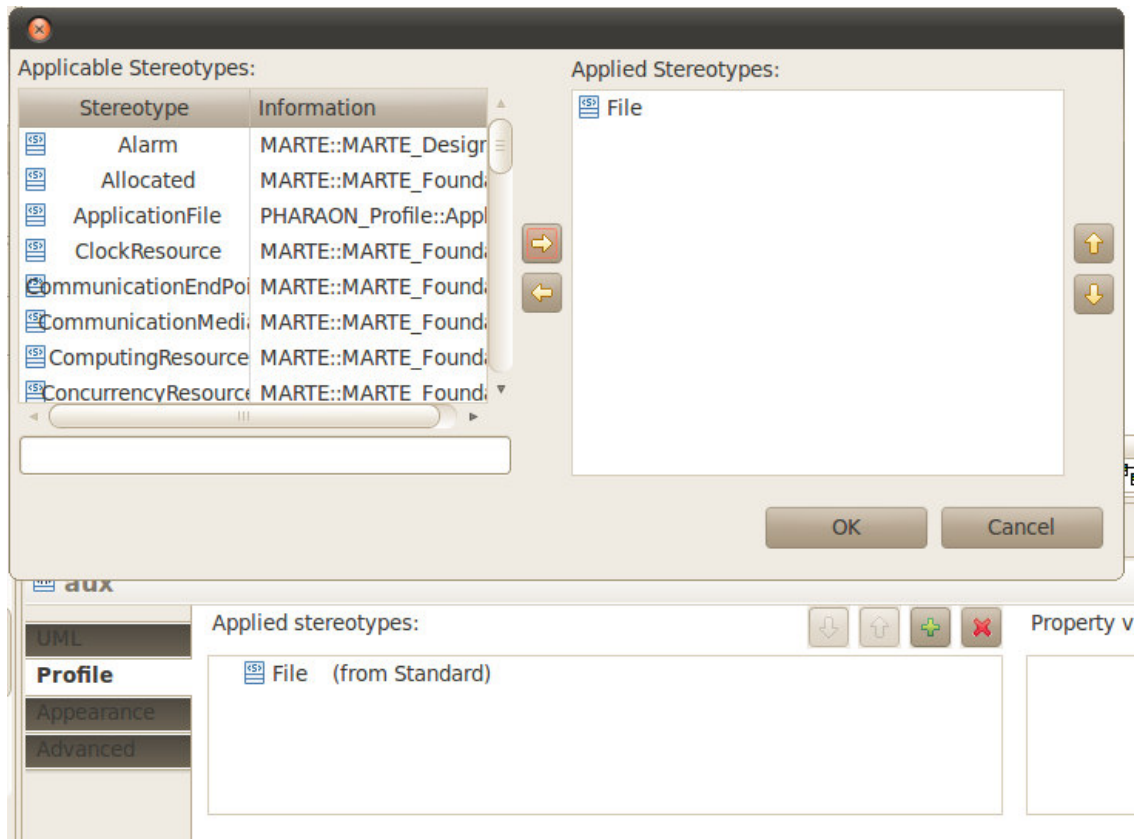


Figure 13 Applying a stereotype

In the model explorer new folders can be created under a view (right click on the View, New Child → Create a new package) and elements can be moved inside the new folder to keep a clean explorer. It makes sense to have one folder per diagram maintaining coherency but it is up to the user (it has no functional effect).

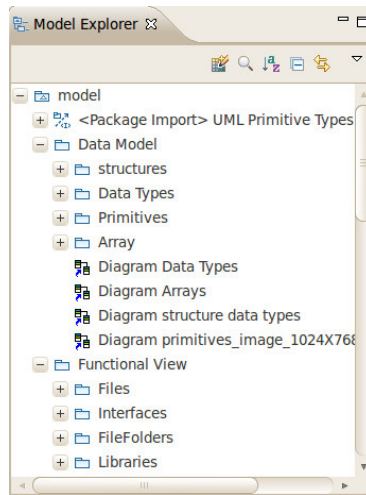


Figure 14 Model Explorer showing the contents of two views (Data Model and Functional View) organized in folders and the marching diagrams (for Data Model view)

Creating the PIM views

Data View

Create the package, rename it and stereotype it as **DataView** as explained in Basic actions above.

Create diagrams to place the different data types (for instance arrays, structures, primitives, enumerations, data types...).

In order to add new elements to the diagram, you can click and drag from the Palette → Nodes the element type required (DataType or PrimitiveType).

Once added to the diagram, it has to be named, stereotyped, and when required (depending on the data type) the attributes set.

Details on data type modelling are found in the modelling reference document [1], Chapter 1

Functional View

Create the package, rename it and stereotype it as **FunctionalView** as explained in

Basic actions above.

Functional View requires the creation of all files with the functional code of the components (plus folders and libraries to complement them) and the interfaces. Normally a diagram and a folder will be created for each of them to keep it organized.

Creating Interfaces

The interfaces group a set of services which define the functionality provided or required by an application component. Every component has interfaces with other components, and each of those interfaces need to be modelled in this view.

Interfaces are created, named and stereotyped as described in

Basic actions above. It can be done from the Palette or from the Model Explorer, New Child list → Create a new Interface. Stereotype it (“clientServerSpecification”) and name it.

Then services need to be added to the interface. Again it can be done from the Model Browser (right click on the interface, New Child list → Create a new Operation) or click on Operation in the Palette and drag it into the Interface box in the diagram. Name the Operation (Properties → UML). No stereotype is applied to the Operation.

For defining the parameters of the service, from the Model Browser right click on the operation of the interface, New Child list → Create a new parameter”. If the service have more than one parameter, the name of the parameter has to be “order:nameParameter” for denoting ordering sequence in the function declaration (Figure 15).

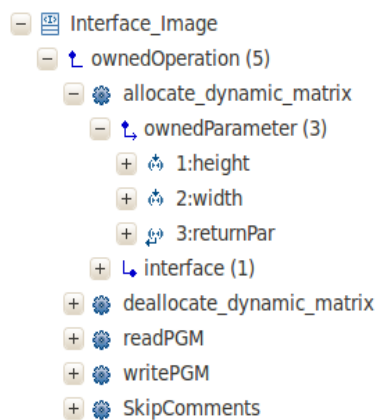


Figure 15 Function parameters

The parameters have to be typed. In the tab “Type”, attached the data type of the parameter (in Figure 16, String).

The tab “Direction” specifies the parameter direction: in out, inout, return (Figure 16).

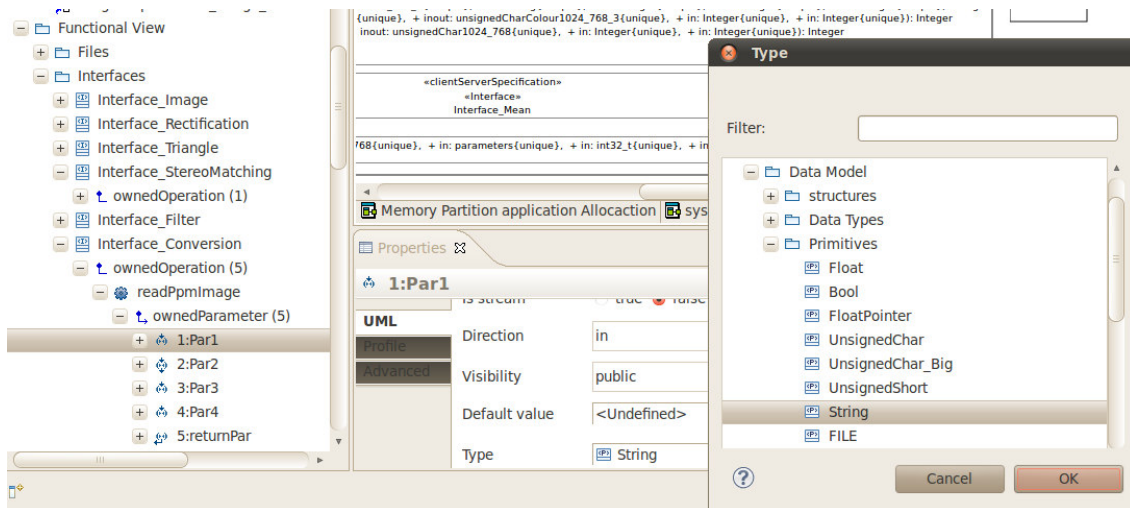


Figure 16 Type the function parameters

below we can see a set of interfaces defined for the communication between the components of a sample application.

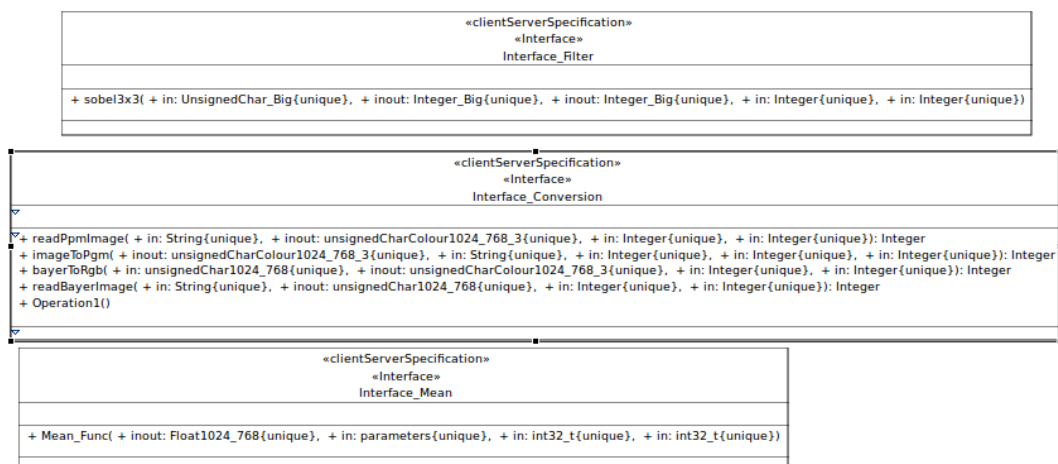


Figure 17 Set of interfaces of the Stereovision components

Files

The files store the implementation source-code of the application components. Files are created, named and stereotyped as described in

Basic actions above. It is done from the Model Explorer, New Child list → Create a new Artifact. Stereotype it (“File”), assign the actual file name with extension and name the element in the Properties → UML tab.

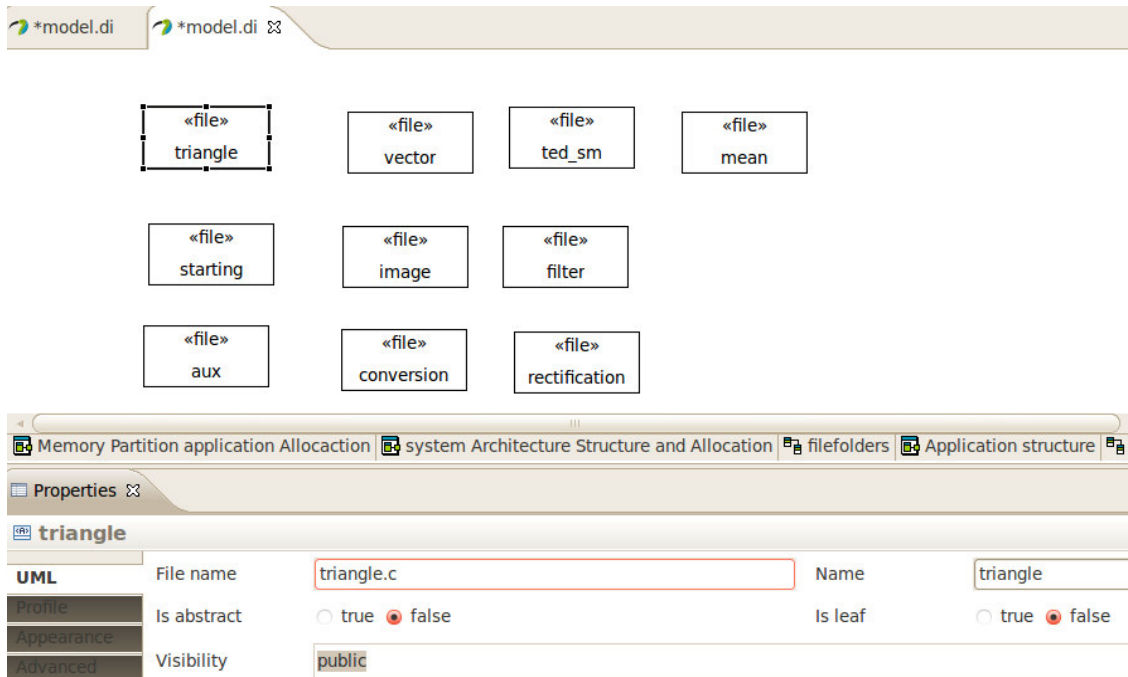


Figure 18 Set of application files and name setting

Details on interface modelling are found in the modelling reference document [1], sections 2.2 and 2.3.

File descriptions may be complemented with more detail by applying the stereotype “ApplicationFile” beside “File”. This allows defining parallelization, target a specific core, and

The function interface defined in the source code must match the model of the SW component. An interface “offered” by a component requires that the top level function of the source code has a matching list of parameters. An interface “required” by a component requires that within the source code of the component there is a call to a function prototype with a matching list of parameters.

Details on interface modelling are found in the modelling reference document [1], Chapter 2.4

few other details.

Application View

Creating Application Components

Components are the basic building blocks of the application model.

Components are created, named and stereotyped as described in

Basic actions above. It can be done from the Palette or from the Model Explorer, New Child list → Create a new Component. Stereotype it (“RtUnit”) and name it.

All components are placed in a dedicated diagram (or several), under Application View (create Class Diagram from the Model Browser). This diagram is only a container of all declared components, with no other system information (as would be connectivity).

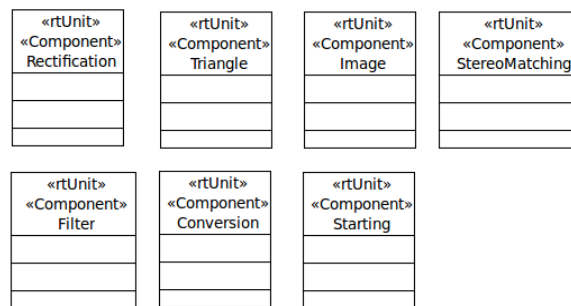


Figure 19 Set of application components

The following attributes of the rtUnit stereotype are relevant:

- *isDynamic* should be true (default).
- *srPoolSize* should be defined by a value >0 (e.g. 10). Default is 0, needs setting.
- *srPoolPolicy* should be *infiniteWait* (default).
- The *isMain* attribute can be set true to denote the main application. Default false.

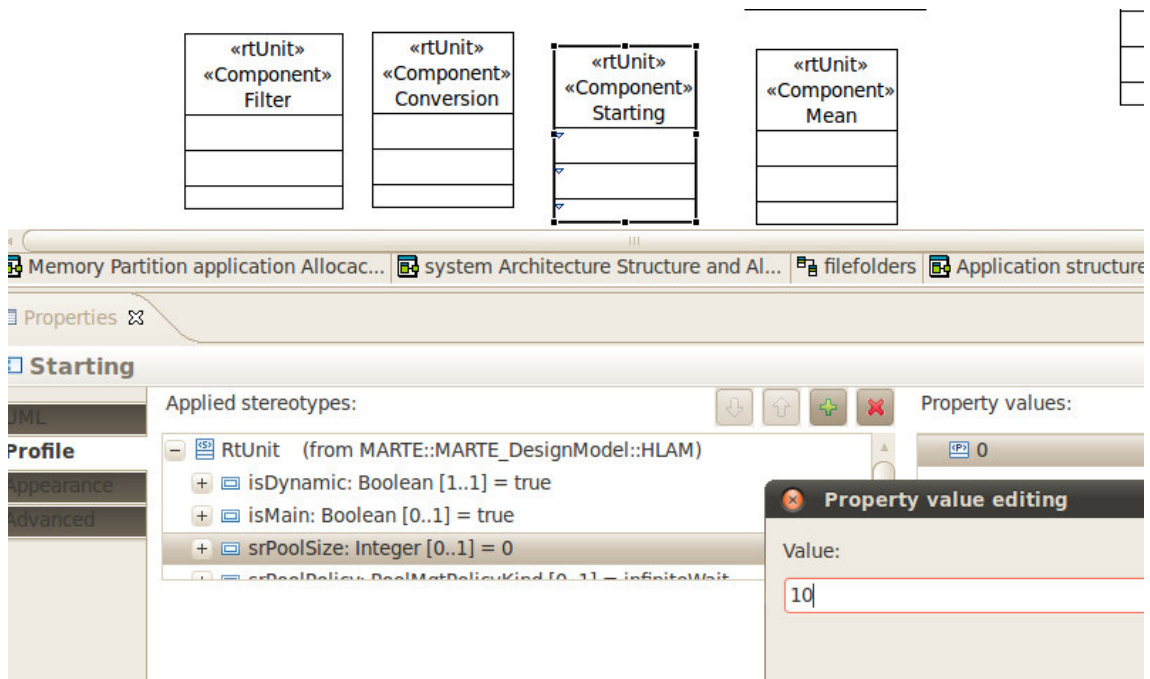


Figure 20 Editing the attribute *srPoolSize*

Application Components-Files Association

Each application component requires the files with the source code to be associated.

The association is done inside a diagram where Components and Files are linked by connectors.

Create a new Class diagram (Model Browser, right click Application View, create new Class Diagram). Then, drag and drop on this Class diagram the Application components and the files that can be picked from the Model Browser.

In order to create the file-component associations: On the palette on the right, select “Abstraction” (Figure 21). With this element selected, click on the model element which represents a file and then click on the model element which represents an application component. Stereotype it as “Allocated”. The same actions are performed to associate FileFolders to components.

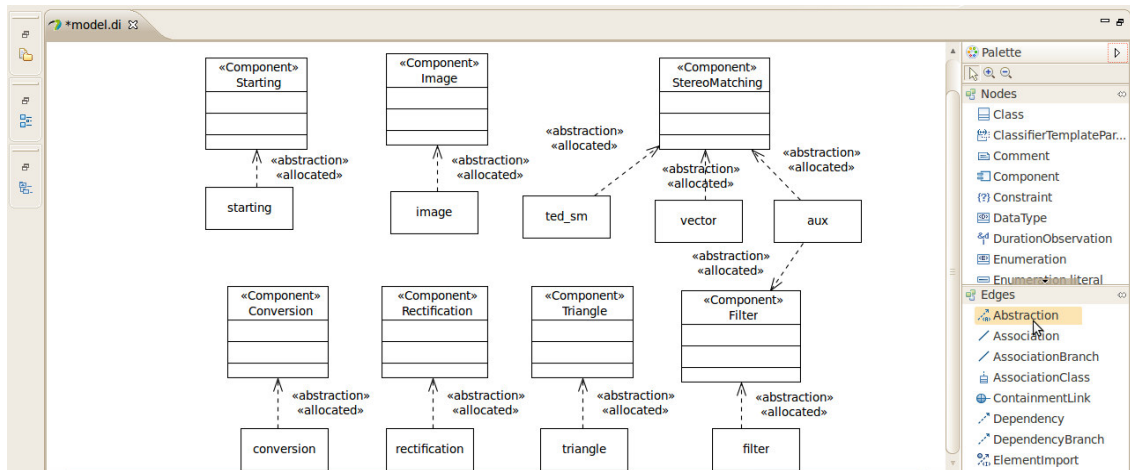


Figure 21 Association files-application components

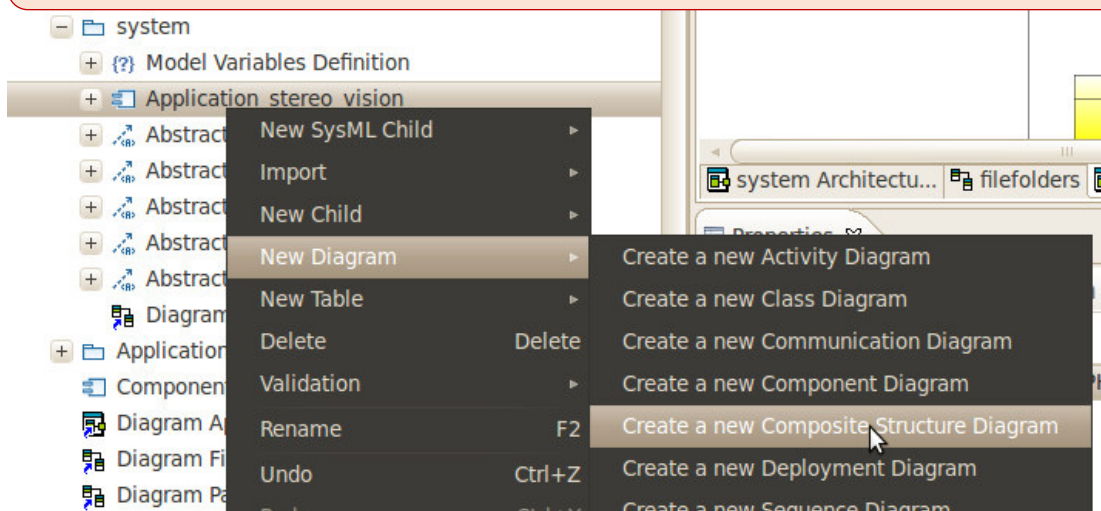
Application Structure

The application structure is composed of instances of application components whose ports are interconnected through channels.

Create a new component which represents the entire System application structure. From the Model Explorer, New Child list → Create a new Component. Stereotype it as “System” and name it. In this case the System component is not placed in a diagram.

Associate a Composite Structure diagram: From Model Explorer, right click on the *System* application structure component. New Diagram → Create a new composite structure diagram.

Details on Files and FileFolders association are found in the modelling reference document [1], sections 4.1.2 and 4.1.3.



On the composite structure diagram (Figure 22) you will get the application structure.

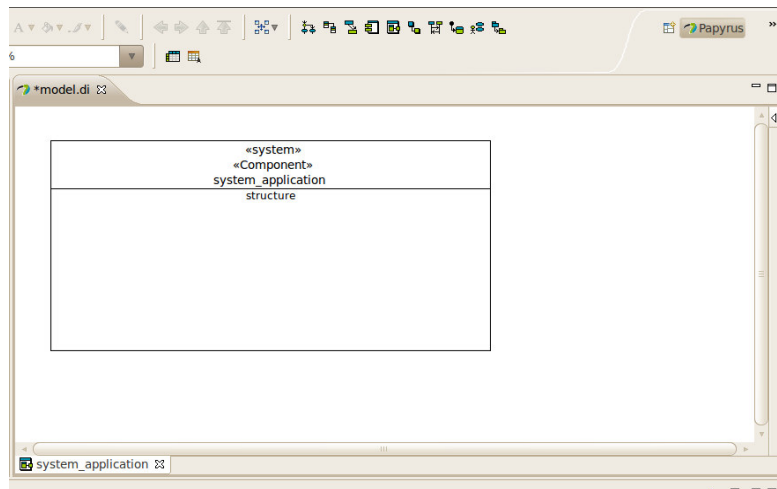


Figure 22 Composite Structure diagram

Create the application instances

Create an instance of an application component: drag-and-drop the application component from the Model Explorer into the *System* application structure component.

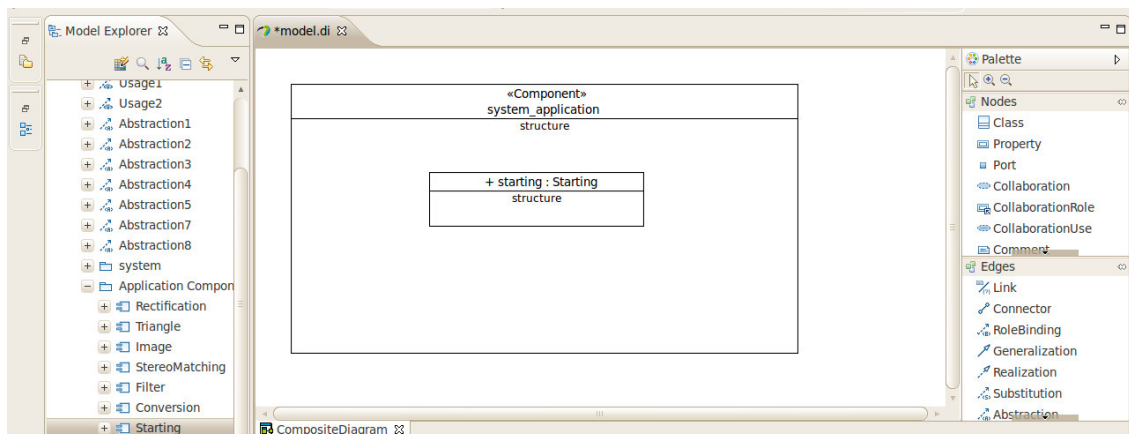


Figure 23 Application Instance

Associate ports to the application instances

The application components are connected to ports. The ports denote the services encapsulated in interfaces that the application component required or provided.

Ports are added to an application component instance in the *System* application structure component. On the palette Nodes, select "Port" and click on the application instance (Figure 24).

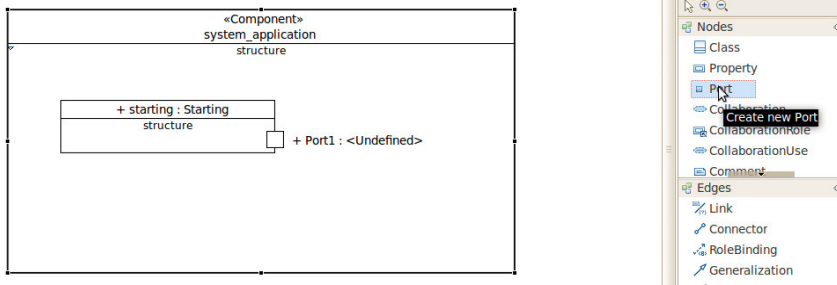
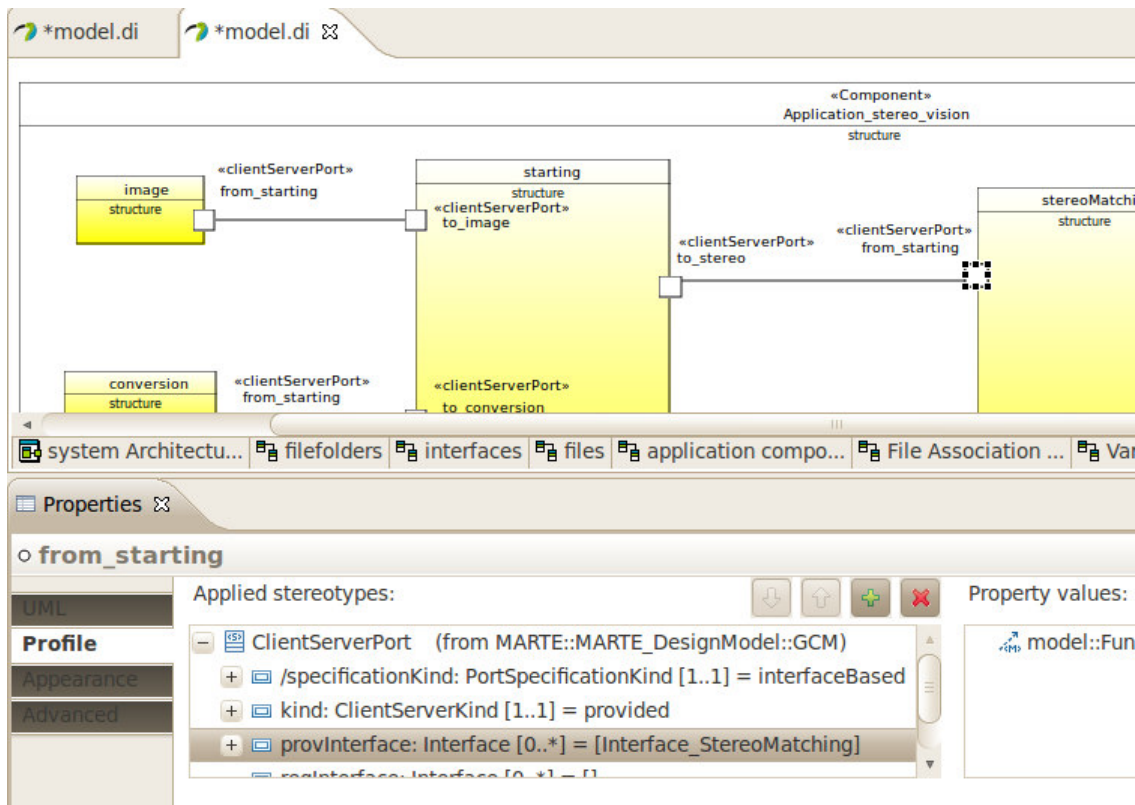


Figure 24 Port on an application Instance

Then, the port should be named and stereotyped as clientServerPort. The attributes that need setting are:

- Kind: ClientServerKind = provided or required
- provInterface: Interface = <The name of the associated interface from Functional View>



Connect the ports: channels

The application components are interconnected through connectors. These connectors may represent channels with well-defined semantics.

The ports are connected using connectors. In the palette, go to “Edges” select “connector” and click in both ports (Figure 25). This connector can represent a channel or a direct service call. In the case the connector is a channel with a well-defined semantics, the UML connector is stereotyped as “Channel”.

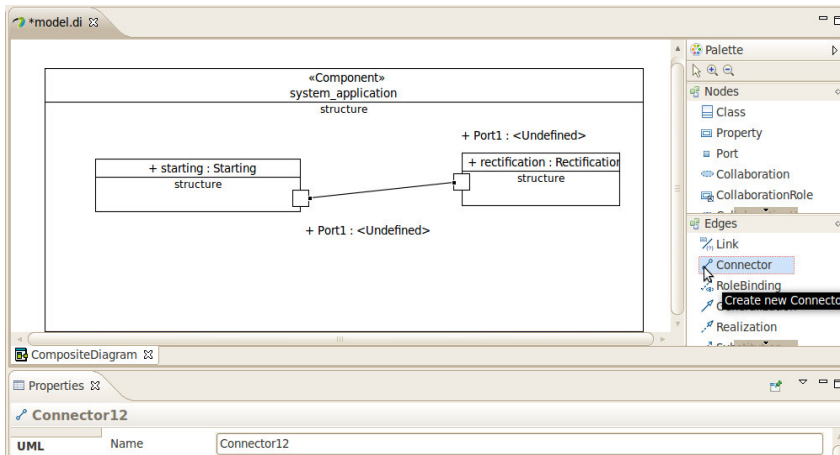


Figure 25 Connection between ports

Channel types

As was previously mentioned, connectors between application components can represent channels. The communication semantics of the channel is captured in channel types. Channels are communication media with a well-defined semantics, according to a set of behavioural characteristics provided by the channel type that specifies the connector. The channel types are captured as Components. From the Model Explorer, New Child list → Create a new Component. Stereotype it and name it. Channel type components require three stereotypes to be modelled: “CommunicationMedia”, “StorageResource” and “ChannelTypeSpecification”.

PDM

The specification of the PDM consists on modelling memory partitions which define executables, the HW resources and SW resources and the HW/SW platform architecture.

Memory Space View

The memory partitions define de executable files that are generated after the final compilation process. Each executable will be compiled for its target core.

Details on Channels and Channel Types are found in the modelling reference document [1], sections 4.1.6 and 3.1.1.

First create a new view and stereotype it as “MemoryAllocView”.

The memory partitions are modelled as Components under the new view, stereotyped as “MemoryPartition”. Then, using instances of these components the structure of the memory partitions is captured, defining the number of executables and the application instances allocation in each of these executables (PSM section).

For defining the memory partitions structure create a *System* component in the same way as the Application Structure. Create a new component which represents the assignment of memory spaces. From the Model Explorer, New Child list → Create a new Component. Stereotype it as “System” and name it. In this case the System component is not placed in a diagram.

Associate a Composite Structure diagram: From Model Explorer, right click on the *System* application structure component. New Diagram → Create a new composite structure diagram.

Then, create the instances which represent the executables of *System* (Figure 26).

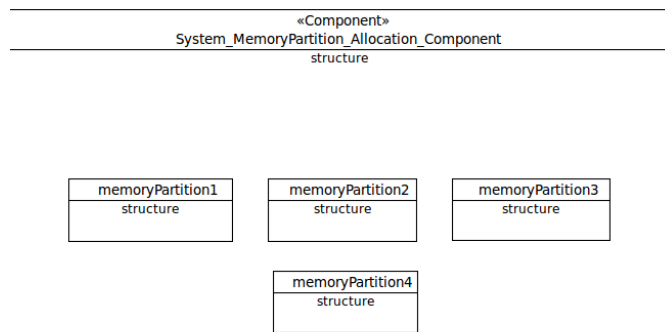


Figure 26 Memory partitions of the System

HW resources view

The HW resources are the components used to describe the platform architecture. The applied stereotype and attributes define the kind of HW resource (CPU, DSP, Memory, Bus,...)

First create a new view and stereotype it as “HWResourcesView”.

The HW resources are modelled as Components under the new view, stereotyped depending of its description. The components are placed in a new class diagram. The most usual resources and the corresponding stereotypes are:

- General Processor: “HwProcessor”
- ISA Processor: “HwISA” (includes DSP, GPU, co-processor like ARM NEON)
- Processor Cache: “caches”
- Network: “HwMedia”
- Network Interface: “HwEndPoint”
- I/O system device: “Hwl_O”

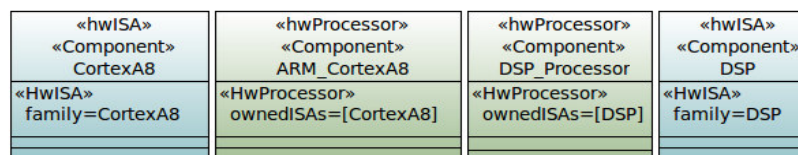


Figure 27 HW platform resources

SW Platform View

The SW Platform defines the operating systems available in the HW/SW platform. First create a new view and stereotype it as “SWPlatformView”.

The SW resources are modelled as Components under the new view, stereotyped as “OS”. The components are placed in a new class diagram.

HW/SW platform architecture

The HW/SW platform architecture is modelled as *System* component. Then, using instances of HW and SW resources components the platform architecture is captured. This architecture is independent of the application. Later (PSM section) the memory partitions from PIM are allocated to the resources defined in this platform.

Create a new component which represents the entire HW/SW platform. From the Model Explorer, New Child list → Create a new Component. Stereotype it as “System” and name it. In this case the System component is not placed in a diagram.

Associate a Composite Structure diagram: From Model Explorer, right click on the *System* application structure component. New Diagram → Create a new composite structure diagram.

Then, create the instances of HW and SW resource for capturing the platform architecture (Figure 28).

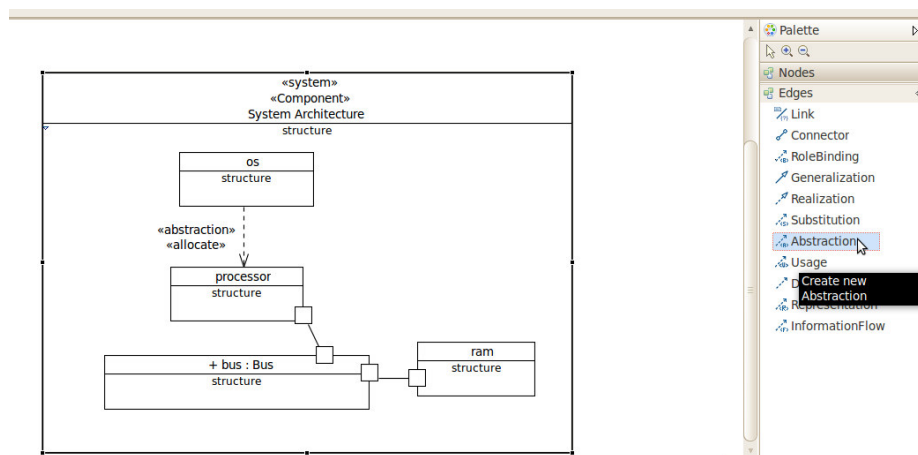


Figure 28 HW/SW platform

For denoting which processor an OS manages, on the right palette, on the tab “Edges”, select an “abstraction” (Figure 28). Then, draw the abstraction from the OS instance to the processor instance. Then, this abstraction is stereotyped as “Allocation”.

The HW instances are connected through ports and connectors. On the palette, go to the “Nodes” tab, select “Port” and right click on the HW instance. For connecting the HW resources, on the palette, go to the “Edges” tab, select “Connector” and right click on the ports of the HW instances to connect.

PSM

The specification of the PSM consists on specifying the different allocations of the system. These allocations are application components-memory partitions and memory partitions-HW/SW platform resources.

Allocation application-memory partitions

For implementing the allocation application-memory, first of all the System application structure component and the Memory Partitions structure have to be associated.

This relation is a UML generalization.

Create a UML class diagram. In the Model Explorer window, right click on the desired view, then New Diagram → Create a new Class Diagram and give it a meaningful name.

Drag-and-drop the *System* application structure component and *System* memory-partition structure component (Figure 29).

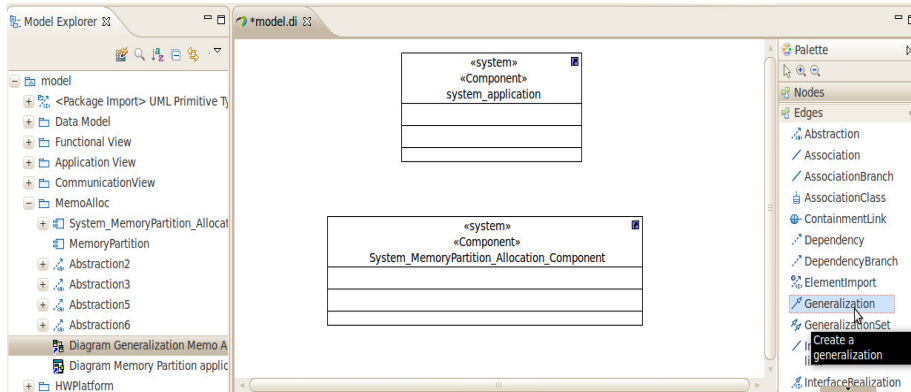


Figure 29 System components of application Structure component and memory-partition structure component

On the palette, go to the “Edges” tab, select “Generalization” relation and click on the System application structure component till the System memory-partition structure component (Figure 30).

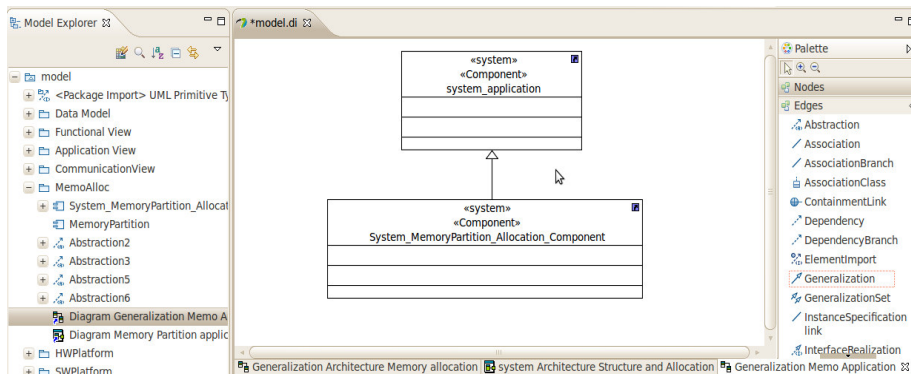


Figure 30 Generalization between the application Structure component and the memory-partition structure component

This *System* structure components generalization enables capturing the mapping of the application into the memory partitions.

Go to the Composite structure diagrams of the *System* memory-partition structure component. Click with the left button on *System* component. Pulse “F4” and the window that Figure 31 shows. Select the application instances.

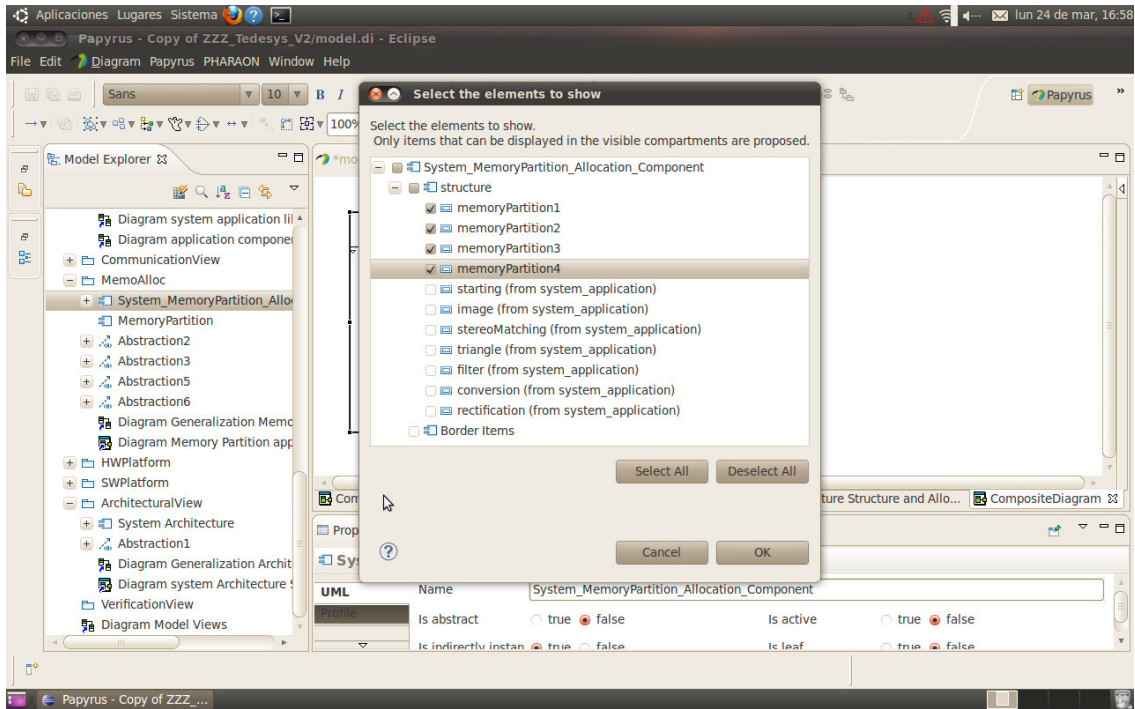


Figure 31 Window after pulsing “F4” for selecting the application components

The result is show in Figure 32.

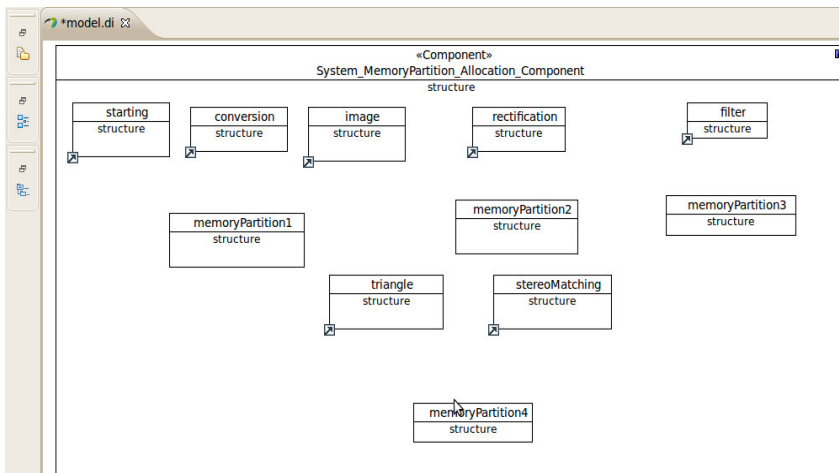


Figure 32 Application component and memory partitions

After that, the application components have to be associated to the memory partitions.

On the palette, on the tab “Edges”, select an abstraction (Figure 33). Then, draw the abstraction from the application component to the memory partition. Then, this abstraction is specified and stereotyped as “allocate”.

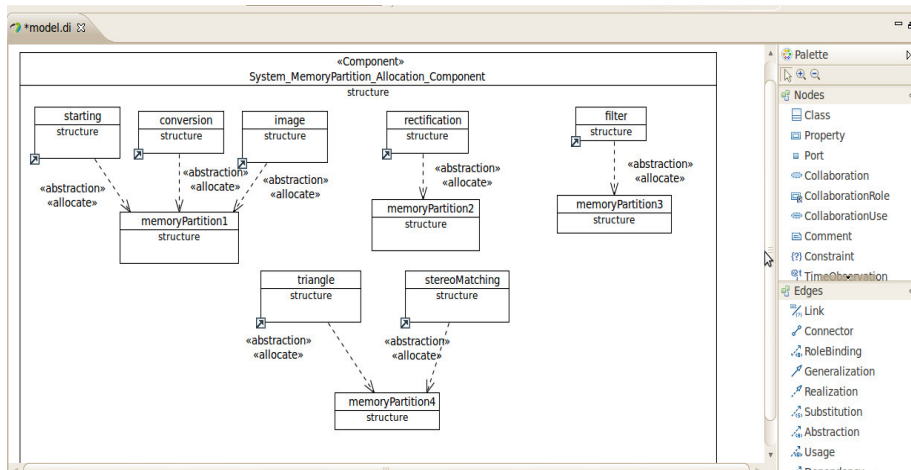


Figure 33 Application component-memory partition allocation

Allocation memory partitions-HW/SW resources

For implementing the allocation memory partition-HW/SW platform the *System* structure components of the memory partitions and the architecture have to be associated.

This relation is a UML generalization.

Create a UML class diagram. In the Model Explorer window, right click on the desired view, then New Diagram → Create a new Class Diagram and give it a meaningful name.

Drag-and-drop the memory partitions and *System* memory-partition structure component (Figure 29).

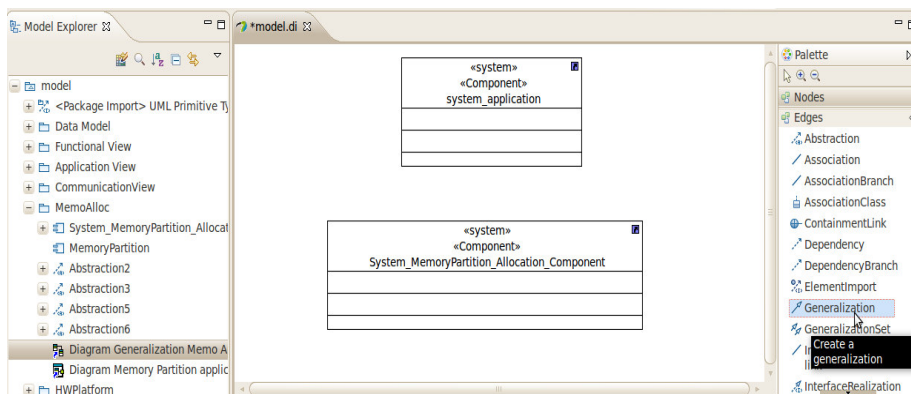


Figure 34 System components of application Structure component and memory-partition structure component

On the palette, go to the “Edges” tab, select “Generalization” relation and click on the System application structure component till the System memory-partition structure component (Figure 30).

On the palette, on the tab “Edges”, select an abstraction (Figure 33). Then, draw the abstraction from the application component to the memory partition. Then, this abstraction is specified and stereotyped as “allocate” (Figure 35).

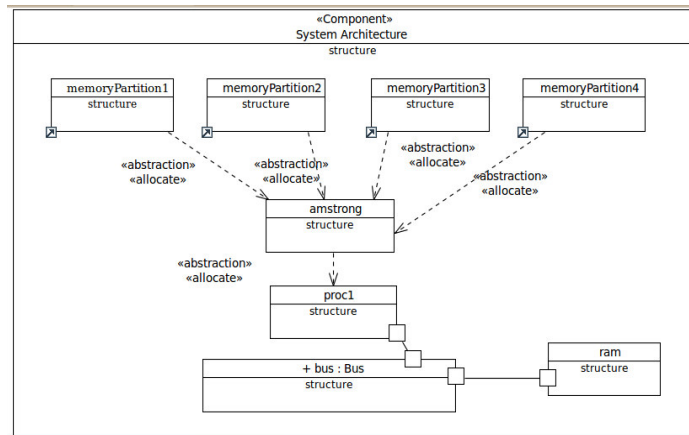


Figure 35 Memory partition architecture platform allocation

Synthesis

Once the complete platform model has been created, synthesys is a matter of few minutes and only three steps.

Always from “PHARAON” tab in ELIPSE:

1. XML Generation

PHARAON → XML Generation → XML PIM Generation

PHARAON → XML Generation → XML PDM PSM Generation

2. WrapperGeneration

PHARAON → Wrapper Generation

3. Makefile generation

PHARAON → Cross Execution → Makefile generation

4. Compilation

PHARAON → Cross Execution → Compilation

The resulting binary files ready for uploading into the host platform will be found under model/outputmodelgeneration/implementation/bin.

References

- [1] UML/MARTE methodology for Heterogeneous Systems design, April 2014.
- [2] C. Szyperski, Component Software: Beyond Object-Oriented Programming, 2nd ed. Addison-Wesley Professional, 2002.
- [3] D. C. Schmidt, "Model-driven Engineering" IEEE Computer, vol. 39 no. 2, pp. 25-31, 2006.
- [4] K. Yamashita. (2010). "Possibility of ESL: A software centric system design for multicore SoC in the upstream phase", Design Automation Conference (ASP-DAC), Proc. of the 15th Asia and South Pacific. pp. 805 - 808.
- [5] OMG: "UML Profile for MARTE", www.omgarte.org.
- [6] www.papyrusuml.org