Extension of the UML/MARTE Network modelling methodology

In Contrex FP7 project

Microelectronics Engineering Group
of the
University of Cantabria

Authors: F. Herrera, P. Peñil, E. Villar

Date: 2015, October 8th

Content

1 Content

This document presents an extension of the CONTREX UML/MARTE network modelling methodology. This document is complementary to the documentation generated so far in CONTREX regarding the definition of a modelling methodology for embedded systems and embedded distributed systems [1] and to the related publications [6]-[12]. The modelling methodology in turn relies on the CONTREX metamodel [2].

As was noticed in D2.2.1 [1], the CONTREX UML/MARTE methodology is built-up on top of previous UML/MARTE modelling methodologies (COMPLEX [3] and PHARAON [4]). These methodologies supported embedded system modelling for design space exploration and for software synthesis.

In CONTREX, the modelling methodology is enhanced to support key aspects in the modelling and design of mixed-criticality systems (MCS) and systems-of-systems (MCSoS) [5]. For it, D2.1.1 introduced criticality in the metamodel. D2.2.1 introduced the mechanisms to describe criticality within the model. In CONTREX, criticality has a wide meaning, in the sense that criticality can be applied to different modelling elements (e.g. application components, platform resources and extra-functional properties) and be interpreted according to the context. D2.2.1 also introduced improvements, such as contract modelling and polishing of the description of the design space to better exploit MARTE.

D2.2.1 also slightly introduced network modelling. Network modelling is a need to enable the methodology to tackle the modelling of embedded distributed systems. As will be shown, the CONTREX network modelling methodologies turns around the network "node", as a fundamental component of the system-of-systems. The CONTREX network modelling methodology enables a rich variety of modelling possibilities of the node. It is required for a compact and coherent modelling of the "computing continuum" present in real networks, which can range from small sensors or actuators to big data centres.

This documents extends the work in [1] and in [6]-[12] in order to support a "multi-level modelling approach" and a more flexible and automated DSE of distributed embedded systems. The presented extension also enables a more homogeneous integration with previous advances of the Embedded Systems Group of the University of Cantabria on the UML/MARTE-based modelling and design of embedded systems [3][4]. Section 2 provides an introductory background for a deeper understanding of the contribution of the extension proposed. Section 3 literally inserts some of the previous work and results for reader convenience. Section 4 introduces the overall network methodology under the extended perspective. Remaining sections are devoted to introduce the specific features (with emphasis on the extensions).

2 Background

The CONTREX modelling methodology has considered a bunch of relevant work which has already addressed network modelling from UML, and MARTE.

Edalab and U.Verona have developed a methodology for UML/MARTE modelling and estimation of packet-based networks, mostly focusing on wireless networks [6][7][8].

The approach supports the automated generation from UML/MARTE of a SystemC-based model (relying on the SCNSL library). The simulation of this model enables the QoS analysis of the network performance, i.e. how the network performs under the load conditions imposed by the application tasks and the network architecture.

Focusing only on modelling, the methodology supports the description of the network architecture, by relying on *nodes* and *abstract channels*. Nodes and abstract channels enable high-level modelling of network resources. Their attributes reflect computation and communication capabilities respectively. In addition, the methodology supports the modelling of the distributed application by relying on *tasks* and *dataflows*. Tasks are mapped on nodes. Dataflows reflect task-to-task communication without a necessary match to a fixed abstract channel or path.

The methodology also introduces the concept of *zones* and *contiguity*. These concepts enable to group the nodes under zones with the same propagation conditions. The concept of contiguity enables the modelling of the *Resistance* between zones. In this methodology, the Resistance enables a high-level modelling of the spoil of the transmission attributes for those abstract channels crossing zones.

In [8], the [6][7] modelling methodology was extended to support the design space exploration of both HW/SW architectures and network architectures, and to obtain out of it a simulation model, relying on SCoPE+ (for node simulation) and SCNSL (for SystemC-based simulation of the network. In [8], the application tasks are described through UML activity diagrams (to enable the modelling of the applications mapped to the node).

In [9] the modelling approach basing the work published in [8] is detailed. An important difference with regard works in [6][7] is the capability to describe the internal HW architecture of the node. In fact, the node is understood as a HW computation resource with networking capabilities, i.e. with a network interface. For such a description of the node, the component-based approach from [3][4] is adopted. The internal architecture of the node has to contain an instance of a network interface component. This view of the node enables the modelling of distributed OS and their mapping to one or more nodes.

In [10] for the modelling in UML/MARTE of wireless sensor networks (WSN) was present. The primary goals such a methodology are to enable accurate estimations based on the detailed modelling of node architectures, and of their deployed SW. Such accuracy shall enable analysis of the WSN for better optimization, especially for power consumption, a main limiting factor in WSN design; and for realistic analysis of WSN behavior under attack conditions, which typically leads node SW to corner cases. Complementary to these objectives, the methodology also makes a proposal for the modelling of the environment agents which that attack the network introducing noise and traffic.

The modelling methodology (being detailed in [11]) supports separation of concerns by means of views.

Some views support the detailed description of the node internals application, SW architecture, HW architecture, etc. In contrast to the approach in [8], where the node comprises a set of HW resources, including at least a CPU and a network interface, in [10][11], the node is modelled as a complete embedded system, which thus comprises, as well as the HW resources, also the platform SW (typically, the RTOS) and application software. This enables the description of the embedded distributed system as a system-of-systems.

Other views are devoted to the description of the network. Specifically, one view covers the declaration of the different types of nodes, while other view enables their instantiation and interconnection to describe the network architecture.

As the methodology in [8][9] , the [10][11] methodology, uses network interfaces. It also uses battery elements (to model energy capacity of the node) and sensor devices.

The aforementioned work has meant an evident advance on network modelling in MARTE. However, further features are still required to cover the CONTREX modellings needs, driven by the modelling of complex real network in the context of MCS design.

First, there is a lack of a "multi-level approach". Each of the aforementioned methodologies cover a specific perspective of the node. That is, [6][7] provide and abstract view, [8][9] a view of a node as a HW resource, and [10][11] as a complete system. However, a real network will in general reflect a "computing spectrum". Nodes´ computation capabilities can range from a simple amount of logic, to large servers. Similarly their respective functionalities range much in size and complexity. Some nodes will be "closed" systems, in the sense that they do not admit further application SW mapping, while others can admit the migration of further functionality. Moreover, thinking in terms of modelling needs, the designer might want to model in detail some nodes, while not others. There are several reasons for it, e.g. because there is no detailed information about the internal architecture of the node, because such a detail is not required for yielding the accuracy, while the modelling and/or the associated simulation effort to estimate the performance of a networked system were every node is modelled in whole details is not expendable. Therefore, in general, each node requires its own way of modelling, suitable to the modeller and designer needs.

The aforementioned methodologies are not sufficiently flexible to specify a design space. Specifically, regarding the way mappings are expressed in the network modelling. In the aforementioned methodologies tasks-to-node mappings are fixed, either through the task-node association in the deployment diagram [8], either through fixed allocations through associations in a composite diagram. This mechanism is quite inefficient in exploration, since it involves model edition (and thus the regeneration of the simulation model), which is a time-costly process, for the exploration of alternatives. The same can be said for the modelling of the mapping of nodes to zones.

Additionally, specific support stating how to link to the modelling of the physical environment is required. The design of a control embedded systems (CES) and control embedded distributed systems (CEDS) requires the modelling of the closed-loops between the CES/CDES and the environment. In other words, it requires to support the modelling of a cyber-physical system and of a cyber-physical system-of systems. In the context of the network modelling methodology, it has implications on the node description.

The concepts of zones and contiguity introduced in [6][7] focuses on the modelling of geographical area in terms of effects on the attributes on the communication resources. However, a more generic and flexible approach is possible, where other types of zones are supported, to reflect geographical areas where certain physical parameter is constant (or in certain bounds), and/or which enables a specific type of analysis. In such a context, different zone types can be present in the model, and a node can belong to one zone of each type.

3 Previous Results in CONTREX

As was mentioned, [9] is a work-in progress in the context of CONTREX. This work in turn already relies on other previous results which are reminded in the following sub-sections.

3.1 CONTREX Network Metamodel (CONTREX D2.1.1)

In the following paragraphs, the content of section 5 of D2.1.1 referring to the meta model for network modelling is literally reproduced for convenience, since it is the fundament for the network modelling methodology. This description has to be understood as a "network domain view"[1].

The network domain view provides a generic ontology for the modelling of networks, valid for different languages (at least for those present in CONTREX). Furthermore, it has been done relying on MARTE existing domain concepts, such the extension of new concepts is minimized.

Here (in section 5 of CONTREX D2.1.1) we include modeling elements that are useful to realize the validation of distributed applications deployed on communication resources subject to certain error rate. This allows for the modeling of embedded systems connected through partially reliable networks and a high level characterization of the network.

MARTE is suitable for the low level accounting of resource usage in time, and this is also applicable to the networks when they are schedule with concrete arbitration strategies; but when general purpose networks are used or when not detailed accounting of networking needs is available, a high level characterization of communication needs and available capacities can be used. This scales up well not only to general purposes traditional internet protocols but also to wireless and mobile communication.

This chapter is organized in three sections. The first consider extensions to MARTE that help to model the network topology, and the overheads on processors due to the handling of packets to be sent and received. The second proposes extensions to the modeling of the necessary workloads, both, in communication and computing oriented. The last proposes a complete set of modeling elements to capture specific analysis contexts for the validation of required communication needs deployed on the available platforms.

[1] The MARTE standard systematically introduces a "domain view" for presenting the fundamental modelling concepts, before introducing the actual elements which compose a MARTE sub-profile.

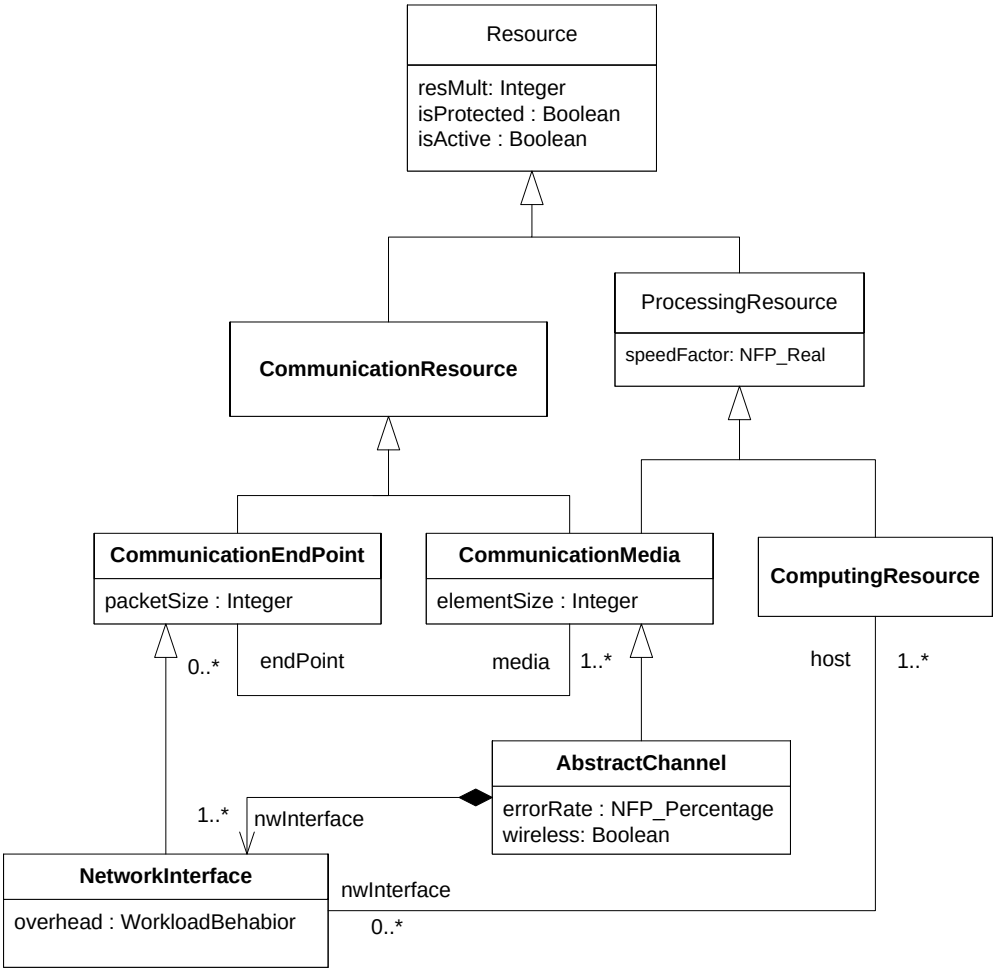
3.1.1 Topology and platform overheads due to communication

Since MARTE profile is devoted to model real time embedded systems, it lacks precise semantics related to networked embedded systems which are mainly for the communication aspects between embedded elements.

Fortunately, and contrary to an often expressed opinion, MARTE had addressed the main elements of such systems and it is not necessary to add new fundamental modeling concepts to MARTE profile. Instead, the work being done in the specification consisted of defining new stereotypes for the communication aspects of embedded systems such as network interfaces.

Therefore, we have introduced new stereotypes to extend the semantics of MARTE profile, stereotypes are:

1. *AbstractChannel,*
2. *NetworkInterface*



- *A **ProcessingResource** generalizes the concepts of CommunicationMedia, ComputingResource, and active DeviceResource. It introduces an element that abstracts the fundamental capability of performing any behavior assigned to the*

active classifiers of the modeled system. Fractions of this capacity are brought to the SchedulableResources that require it.

- *A **CommunicationResource** represents any resource used for communication and may be considered as a collector of communication services. It generalizes the two kinds of communication resources defined, communicationEndpoint and communicationMedia.*

- *A **ComputingResource** represents either virtual or physical processing devices capable of storing and executing program code. Hence, its fundamental service is to compute, what in fact is to change the values of data without changing their location. It is active and protected.*

- *A **CommunicationEndPoint** acts as a terminal for connecting to a communication media, and it is characterized by the size of the packet handled by the endpoint. This size may or may not correspond to the media element size. Concrete services provided by a CommunicationEndPoint include the sending and receiving of data, as well as a notification service able to trigger an activity in the application.*

- *A **CommunicationMedia** represents the means to transport information from one location to another (e.g., message of data). It has as an attribute the size of the elements transmitted; as expected, this definition is related to the resource base clock. For example, if the communication media represents a bus, and the clock is the bus speed, "element size" would be the width of the bus, in bits. If the communication media represents a layering of protocols, "element size" would be the frame size of the uppermost protocol.*

- *A **NetworkInterface** acts as an interface to connect a physical device with a communication media. It has an attribuite WorkloadBehavior which represents a given load of processing flows triggered by external (e.g., environmental events) or internal (e.g., a timer of the communication protocol) stimuli. The processing flows are modeled as a set of related steps that contend for use of processing resources and other shared resources. It may contain the communication protocol agent.*

- *A **Node** represents physical processing devices capable of storing and executing program code. It can be seen as a container of tasks. At the end of the application design flow, nodes will become HW entities with CPU and network interface and tasks will be implemented either as HW components or as SW processes. It can be fixed or mobile node.*

- *An **AbstractChannel** is a generalization of network channels since it contains the physical channel, and all the protocol entities up to level N-1. It has an attribute errorRate which defines the bit error rate of it. It has an attribute wireless to define if it is wire or wireless channel.*

3.1.2 Extensions for modeling general purpose networking workloads

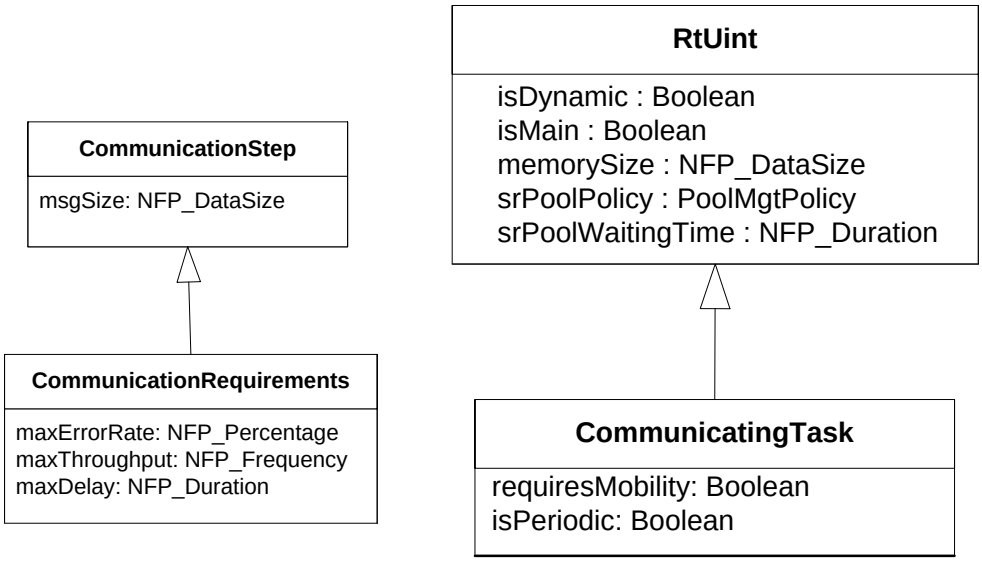
As it was mentioned in section 5.1 that MARTE has provided the main elements for modeling embedded systems but it lacks some semantics related to networked embedded systems.

Therefore, MARTE elements may be extended to compensate such lack. For example, Quality of Service of the communication media between embedded device in terms of, delay, throughput, error rate, is considered as an important feature to measure the performance of such applications.

Therefore, we have introduced new stereotypes to extend the semantics of MARTE profile, stereotypes are:

1- *CommunicationRequirements*

2- *CommunicatingTask*



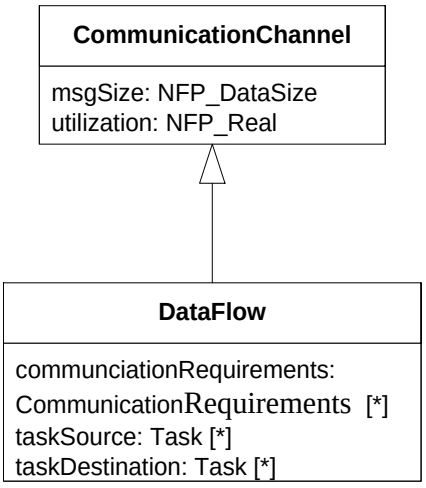
- *A **CommunicationStep** is an operation of sending a message over a CommunicationResource that connects the host of its predecessor Step, to the host of its successor Step.*
- *A **CommunicationRequirements** are the requirements of a data flow to be assigned to an abstract channel and that to establish the communication between two tasks. It has three attribuites, maxErrorRate is the maximum number of errors tolerated by the destination; maxThroughput is the maximum amount of transmitted information in the time unit; maxDelay is the maximum permitted time to deliver data to destination.*

- *A **RtUnit** is real-time unit and it owns at least one schedulable resource but can also have several ones. If its dynamic attribute is set to true, the resources are created dynamically when required. In the other case, the real-time unit has a pool of scheduling resources. When no schedulable resources are available in the possible, the real-time unit may either wait indefinitely for a resource to be released, or wait only a given amount of time (specified by its poolWaitingTime attribute), or dynamically increase its pool of thread to adapt to the demand, or generate an exception. A real-time unit may own behaviors. It also owns a message queue used to store incoming messages. The size of this message queue may be infinite or limited. In the latter case, the queue size is specified by its maxSize attribute. In addition, a real-time unit owns a specific behavior, called operational mode. This behavior takes usually the form of a state-based behavior where states represent a configuration of the real-time unit and transition denotes reconfigurations of the unit.*

- *A **CommunicatingTask** represents a basic functionality of the whole application; it takes some data as input and provides some output. It should be allocated in a Node to perform its operation. It has an attribute named requiresMobility to define its requirement to be allocated in a mobile or fixed node. It can be periodic or aperiodic task and it is specified from isPeriodic attribute.*

3.1.3 Allocation and models for network analysis

In this section we extend MARTE communciationChannel element by a new stereotype for DataFlow to express the communication requirements of the data flow from the communication channel.

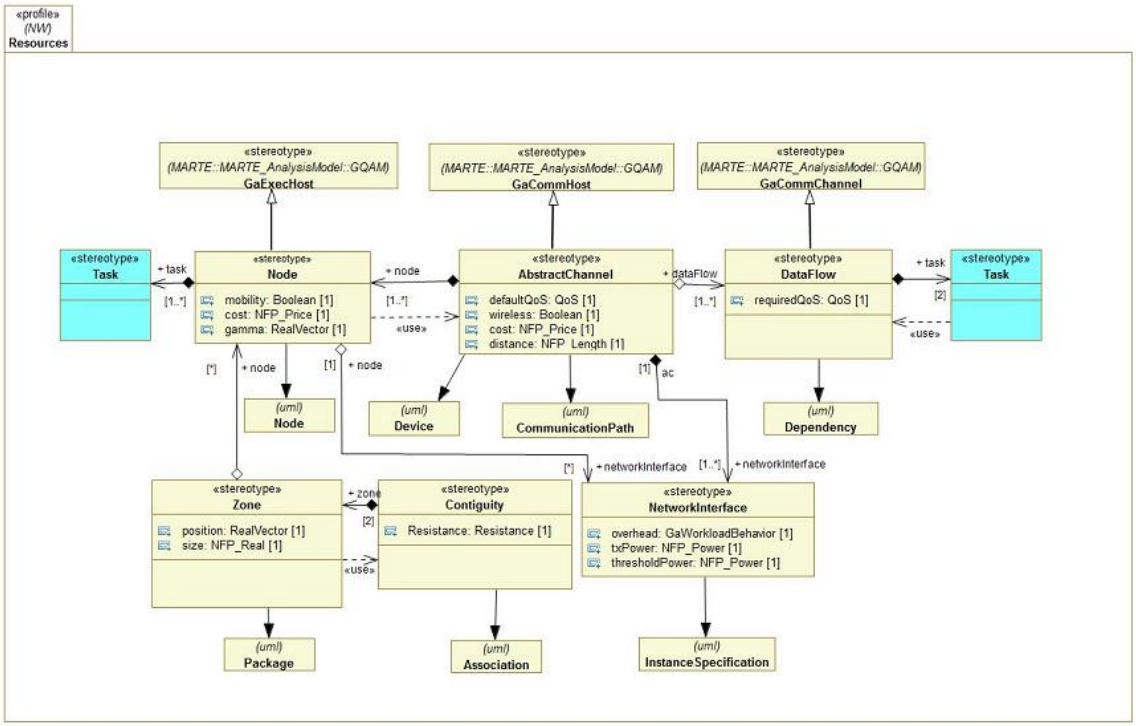


- *A **CommunicationChannel** is logical communications layer connecting SchedulableResources.*

- *A **DataFlow** represents the communication requirements between two tasks; output from a source task (taskSource) is delivered as input to a destination task (taskDestination). It has an attribute communciationRequirements which describes the communication requires to perform the communcaition between two tasks. It should be allocated in an abstractChannel to perform its operations.*

3.2 CONTREX Network Profile

As a result of the metamodeling effort a first profile version has been generated and reported in [12] and in the D2.2.1 CONTREX deliverable. Figure 1 reproduced it here for convenience.



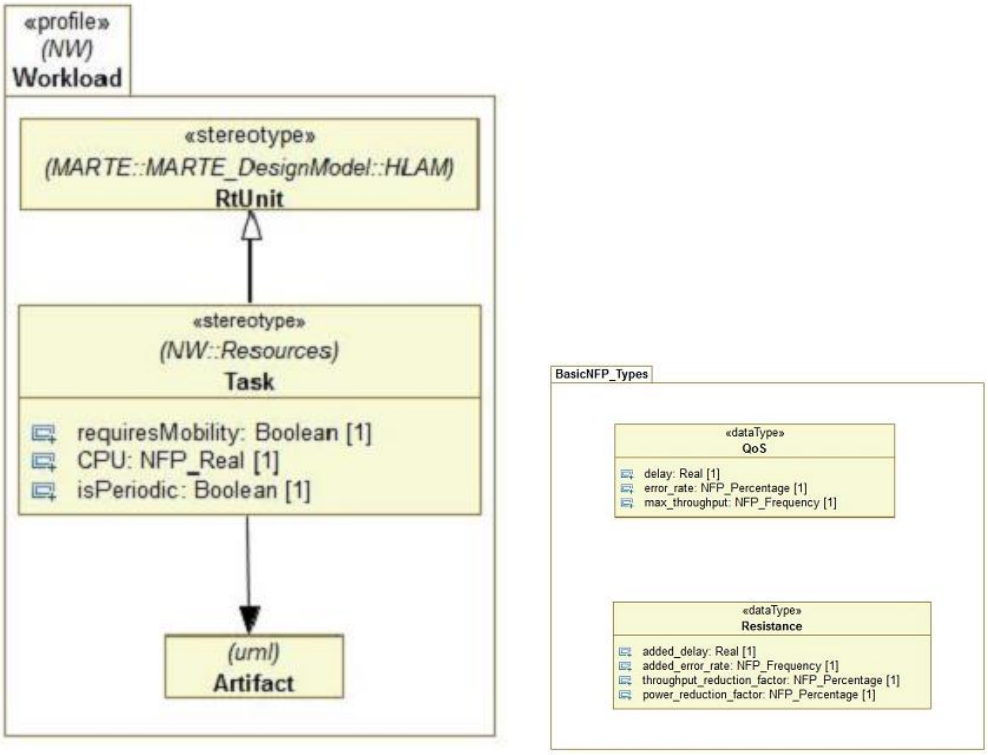


Figure 1 The CONTREX network profile provides additional BasicNFP types and two subprofiles for describing workloads and network resources.

3.3 CONTREX Modelling Methodology (CONTREX D2.2.1)

The D2.2.1 deliverable presented the preliminary version of the modelling methodology. As was mentioned, this document is mostly focused on the modelling of an embedded system and slightly introduced network modelling.

4 Network UML/MARTE modelling methodology

4.1 Purpose and contributions

The CONTREX network modelling methodology fills the gap in the currently available UML/MARTE modelling methodologies, suitable for embedded systems modelling, but yet inefficient for network modelling.

The CONTREX network modelling methodology supports a wide concept of node, a fundamental element in network modelling, to enable the modelling of the "computing spectrum" present in the networks of the IoT era. Being oriented to embedded distributed systems, the methodology covers models where the node is understood as a computational element with network interface capability. Moreover, the methodology supports the modelling of nodes of any type (switches, routers, data center, super-computers) which range different computational and functional capabilities, at different abstraction levels.

Following, a list of contributions of the CONTREX network modelling methodology to the SoA is given, accounting for the aforementioned capability is given:

- It enables a multi-level modelling approach. The approach is multi-level in the sense stated in [5], that is, that the methodology enables modelling network nodes of different types and at different levels of abstraction.

- It enables the flexible mapping and exploration of a distributed applications into a network;

- It establishes the link to model CPS and CPSoS;

- It extends the concept of zones, by supporting several types of them in the same model, and makes the mapping to zones more flexible for exploration.

The CONTREX network modelling methodology is smoothly integrated with the embedded system modelling. A component-based modelling approach and a reduced but efficient set of modelling techniques is homogeneously applied at every modelling level: application, SW platform, HW platform, and network.

By relying on the CONTREX metamodel, the concepts presented in the background methodologies shown in section 2 are covered.

4.2 Modelling elements

The modelling elements to be employed in the network modelling methodology are the ones defined in the CONTREX metamodel.

As a reference, the profile shown in section 3.2 is used. Required extensions of that profile will be reported in place. Most of them are simply profile extensions which do not require an actual extension of the metamodel, keep back compatibility with the modelling approaches proposed in [6][7][8], and which enable a more convenient and flexible UML modelling style.

4.3 Network modelling views

The CONTREX modelling methodology supports two specific views for network modelling: the node view and the network view. The views are specified as UML packages with the <<NodeView>> and <<NetworkView>> stereotypes.

The node view serves to declare the different types of network nodes present in the network mode. The network view is used to capture the network architecture.

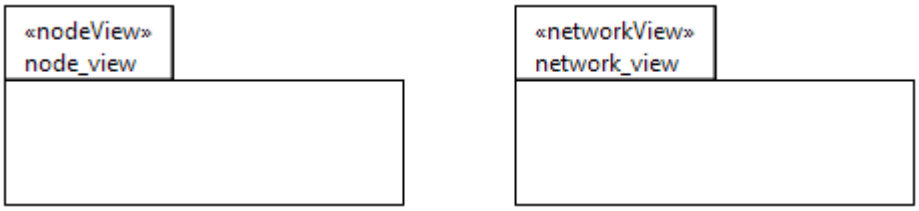


Figure 2 Network and node views.

Node and network views hang from the root of the model, in the same level as other views devoted to the description of an embedded system, such as the application view, the SW Platform view, and the HW resources view.

As reflected in Figure 3, the network model depends on the information captured in the application, SW platform, and HW resources views. It means that, for the completion of the capture of the node view and of the network view, the capture of the application, SW platform, and HW resources views has to be completed first.

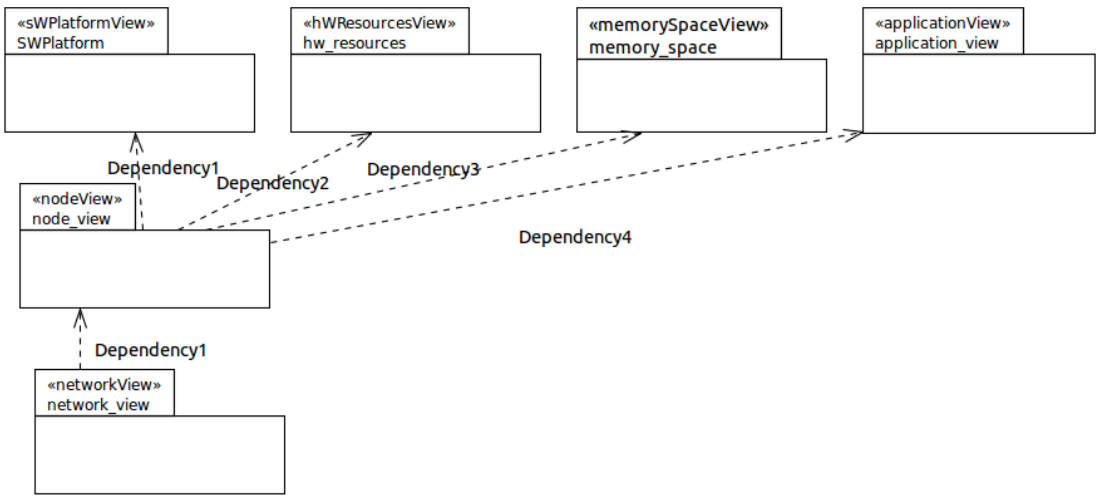


Figure 3 Network and node views depend on the application, SW platform and HW resources view.

However, to the effects of modelling workflow, the concurrent capture of some elements in the different views is possible. For instance, the architecture of the network can be captured upon nodes incompletely captured, i.e. where not all of their attribute values have been fixed.

4.4 Network Architecture

The network architecture is a primary information with relevant impact on the overall performance of either a System-of-Systems or a distributed application. The network architecture consist of the set of node instances present in the network and how they are interconnected. In other words, the network architecture captures the network topology.

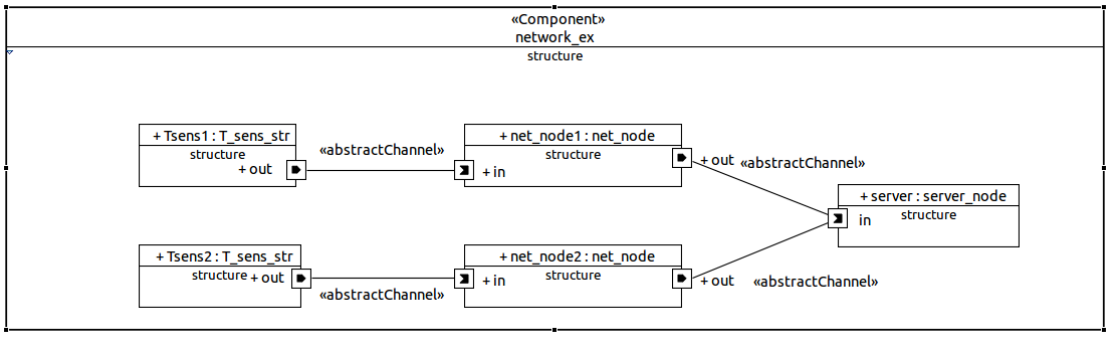


Figure 4 .Example of network architecture. Five network nodes of three node types are instanced in a "network_ex" component located in the network view.

The network architecture is described in the network view, i.e. <<NetworkView>> package.

Within the <<NetworkView>> package, a network component has to be declared. A composite diagram associated to the network component is used to capture the network architecture.

UML properties are used to capture node instances, e.g. "Tsens1" or "net_node1" in Figure 4. The type of node instance is captured by typing the UML property with one of the node components declared in the node view. For instance, "Tsens1" node of type "T_sens_str", and "net_node1" is a node of type "net_node", e. g. to model a sensor and a router node.

Port-to-Port connectors enable to link nodes and model ideal point-to-point connection among network nodes.

Moreover, modelling of "non-ideal" point-to-point communication links is supported by stereotyping the port-to-port connectors as <<AbstractChannel>>, as shown in Figure 4. The <<AbstractChannel>> supports abstract modeling of non-ideal point-to-point transmission characteristics. Specifically, it supports the modeling of:

- Error rate (provided by <<AbstractChannel>>)
- Throughput (Hz) (inherited)

- Capacity (Tx Rate) (inherited)

These characteristics are due to the consideration of *low-level* factors, e.g. type of transmission physical media, material of the wire, distance between access points, etc.

The aforementioned attributes of <<AbstractChannel>> provide a convenient abstraction for data-packet network modelling as long traffic conditions and low-level factors, like distance among nodes, can be considered stable (fixed) along time.

If this is not the case, annotated values can be considered for the initial state of the QoS of the communication links.

A main scenario where data links transmission conditions happens where there are wireless network involved. To tackle an abstract modelling of it, the methodology support stating if the point-to-point link is wireless or not. As was shown in section 3.2, the <<AbstractChannel>> stereotype supports the modelling of an important intrinsic factor, if the channel is wire or wireless. Zones and contiguity enable taking into account mobility while keeping the abstract modelling of node communication characteristics. This is discussed in detail in section 4.8.

Notice also that a network modelling methodology can consider several lower levels of detail in the modelling of communication resources, e.g. to account for the impact of communication protocols, node distances, geometry, etc. However, such a low-level modelling of the underlying communication infrastructure is possible, but it also has an additional, non-negligible modelling and simulation cost.

4.5 Node Modelling

The proposed network modelling methodology provides a rich variety of modelling approaches for network nodes. Specifically, the methodology supports the modelling of different types of nodes at different abstraction levels.

The methodology enables the following abstraction levels in the modelling of a node:

- description of its internal architecture
- description of a behavioural model
- as a deterministic or statistic traffic pattern generation/consumption

Moreover, in the former case, the methodology supports the modelling of a node: as:

- a HW computation resource
- a SW/HW platform, i.e. a HW resource with a SW layer which enables call computation and communication services,

- a complete system, in the sense that the node includes autonomous applications which can cooperate with other nodes thanks to the networking capabilities of the node.

Subsection 4.5.1 presents where and how network nodes are declared. The following subsections show the mechanisms available to describe a node.

Subsections 4.5.2, 4.5.3, 4.5.4, refer to detailed node description mechanisms where its internal architecture is captured. Subsection 4.5.5 points out where the components used to describe the node internal architecture are taken from.

Subsections 4.5.6 and 4.5.7 refer to more abstract ways to model the nodes.

4.5.1 Node declaration

Nodes are declared in the node view, i.e. the <<NodeView>> package. Nodes are declared as components with the <<Node>> stereotype applied.

Figure 5 and Figure 6 show two examples of node declarations.

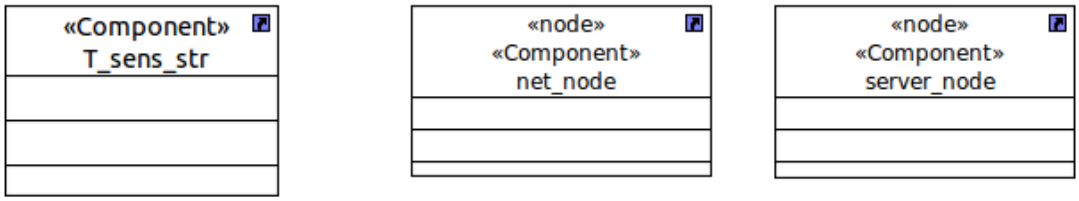


Figure 5 Example of declaration of nodes whose internal structure is described.

Figure 5 shows an example where three types of network nodes (a temperature sensor node, a network node, and a server node) are declared.

Figure 6 shows an example whith more types of nodes are declared. For instance, to declare temperature and light sensor, and heater and cooler actuator nodes, a network (router) node, and a server node.

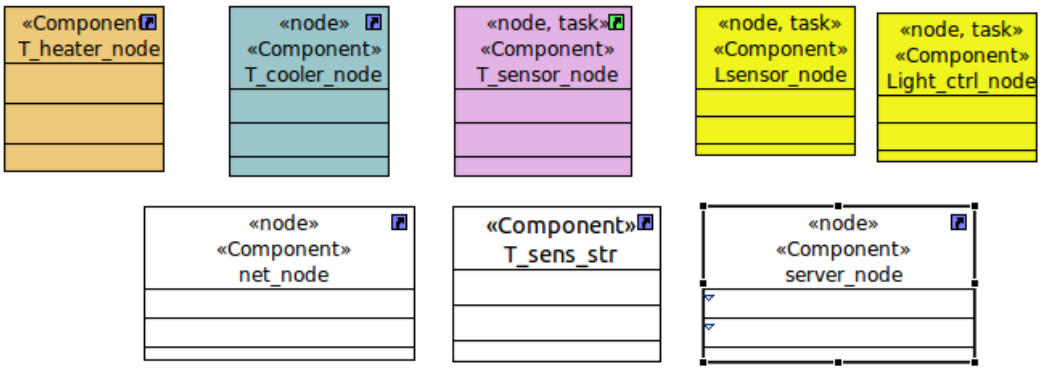


Figure 6. Example of node declarations for the description of a distributed control of temperature and light in a building.

In the Figure 6 node view, some colouring has been used to distinguish node types. It can be useful, e.g. to help the user to identify node types, according to the own user criteria. However, colouring is an UML editor dependent feature, and it is not part of the semantic model.

4.5.2 Description of the node as a HW resource

The methodology supports the modelling of the node as a cluster of HW resources with access to, and that can be accessed from, the network. The node will include at least computational HW resources, and HW resources for accessing the network (network interface). Eventually, it can also contain other type of hardware resources (memories, sensors, actuators, battery, etc).

The description of the node as a HW resource is similar to the description of the HW architecture when a single-system model is developed. That is, a composite diagram is associated to the node and instances of the components declared in the HW resources view are done and interconnected through port-to-port connectors.

However, there are specific considerations in the description of nodes as a HW resource:

- Since the node view contains several nodes in general, and the internal architecture of the node can be described, a model can contain several HW architectures (which is not the case of the single-system scenario in the modelling methodology[2]).

- The HW architecture of the node has to include an instance of a network interface component instance.

- The previous rule requires, in turn, the declaration of at least one network interface component within the hardware resources view.

- The node component has to have at least one flow port, captured as a <<PortFlow>> port, which reflects a logical packet traffic interface between the node and the rest of the network[3].

- The aforementioned flow port has to be associated to a network interface.

---

[2] Here it is convenient to remind that in the development of a single-system model, e.g. a MPSoC model, only one <<System>> component within the architectural view is employed and allowed to capture system internal architecture. In contrast, when a network model is developed, the user in general will want to be capable to model several sub-systems interconnected among them. The methodology supports it through the capability of declaring several nodes in the node view. Each of those nodes support the description of its internal architecture, as it was done for the system component in the single-system modelling approach. The result is that the user can develop a System-of-Systems (SoS) model with a precise description of each node involved

[3] Notice that this flow port reflects neither a high-level functional interface, nor a low-level interface, i.e. a bit-level or physical description of the network traffic

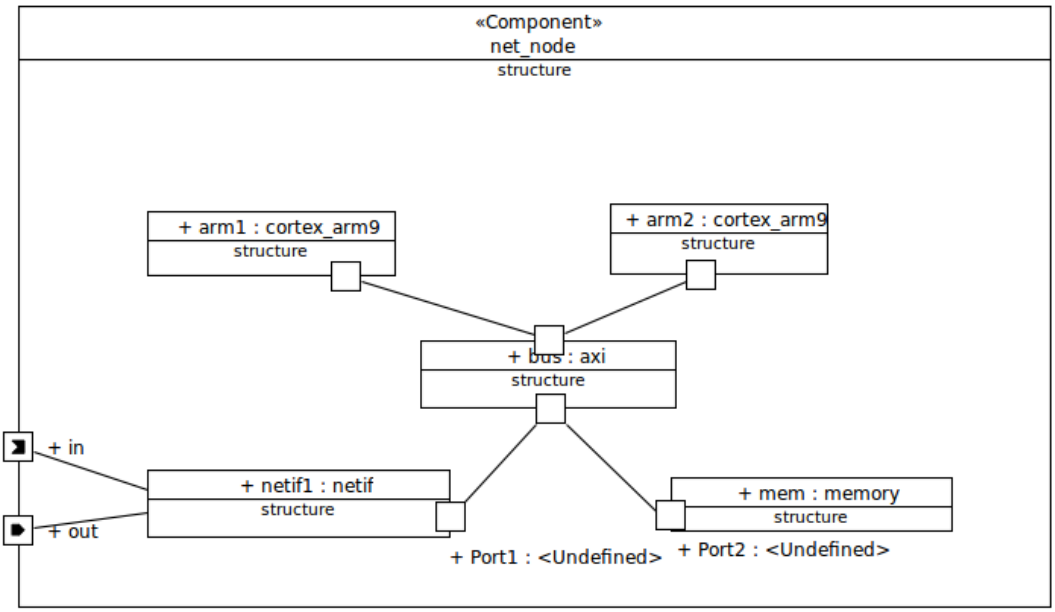


Figure 7. Example of the description of a node as a cluster of HW computational and network resources.

Figure 7 shows an example of the description of the internal architecture of a node as a HW resource.

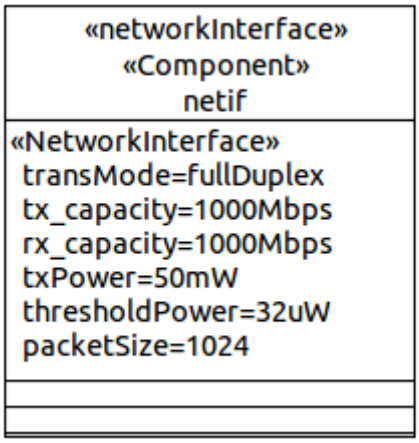


Figure 8. A network interface component declared in the node view.

The network interface component instance (“netif1”) is captured as a UML component with the <<NetworkInterface>> stereotype applied.

The stereotype provides key attributes in the description of the transmission and reception capabilities in the connection of the node to the network. Some of these attributes (thresholdPower, txPower, packetsize) are supported by metamodel shown in

section. Other attributes (tx_capacity, rx_capacity, transMode) are supported after the extension illustrated in Figure 9[4].

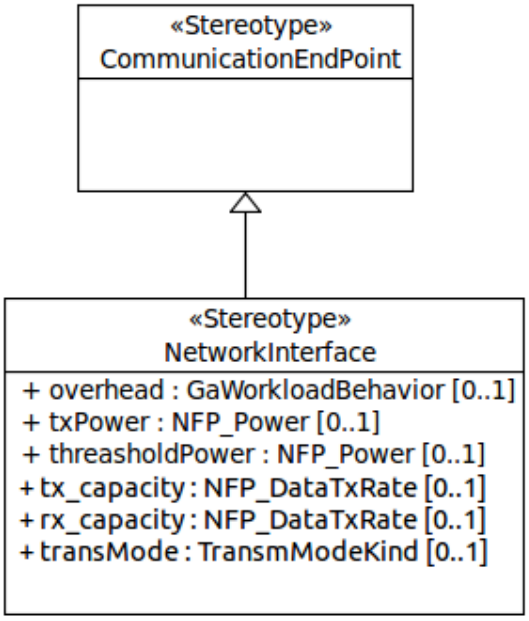


Figure 9. The <<NetworkInterface>> also has a capacity attribute.

The model can leave these attributes unset. In such a case, if the attributes of the abstract channel are settled, only they are considered, which enables a very synthetic network modelling.

Moreover, in the more general case, the user can state the attributes of the abstract channel and of the network interfaces of the transmitter and receiver nodes, which enables a more accurate modelling.

Specifically, if the capacity attributes are stated also in the network interfaces, then the more restrictive capacity attribute either at the network interface side, or at the abstract channel side applies. Following, some modeling cases illustrate the semantics:

- The node-to-node connection is ideal (no <<AbstractChannel>> stereotype applied), and the network interfaces at the transmitter and receiver nodes have no settled attributes. This means an ideal point-to-point link between the nodes.

- The node-to-node connection is ideal (no <<AbstractChannel>> stereotype applied), and the network interface of the transmitter node has a settled attribute, e.g. capacity=1000Mbps, while the network interface at the receiver side has no capacity attribute settled. Then, the capacity of the transmitter network interface

[4] Other extension possibilities have been assessed and finally discarded, which is documented in section 6.2 of this report.

defines the maximum speed of the point-to-point node connection, i.e. 1000Mbps.

- The node-to-node connection has the <<AbstractChannel>> stereotype applied, with settled attributes, e.g. capacity=1Mbps, and the transmitter network interface has a settled attribute too, e.g. capacity=1000Mbps, while the receiver network interface has not capacity attribute settled. Then, the most restrictive capacity defines the maximum speed of the point-to-point node connection, 1Mbps in this case. If, to the contrary, the transmitter network interface had capacity=256K, then 256Kbps would be the resulting point-to-point capacity.

4.5.3 Description of the node as a SW/HW platform

The methodology supports the modelling of the node as a SW/HW platform with access to, and that can be accessed from, the network. That is, as well as a set of HW resources (at least computational and network resources), the node also contains a set of SW platform resources, i.e. an RTOS and eventually the required drivers.

The capture of the internal architecture of a node as a SW/HW platform follows the same rules as for the capture of a node as a HW platform, plus the inclusion of at least the following elements:

- An operative system instance
- The allocations of every operative system instance to at least one processing element of the HW architecture of the node

Figure 10 provides an example of the internal architecture description of a structured node, specifically of the "server_node" instanced in the network architecture example show in Figure 4.

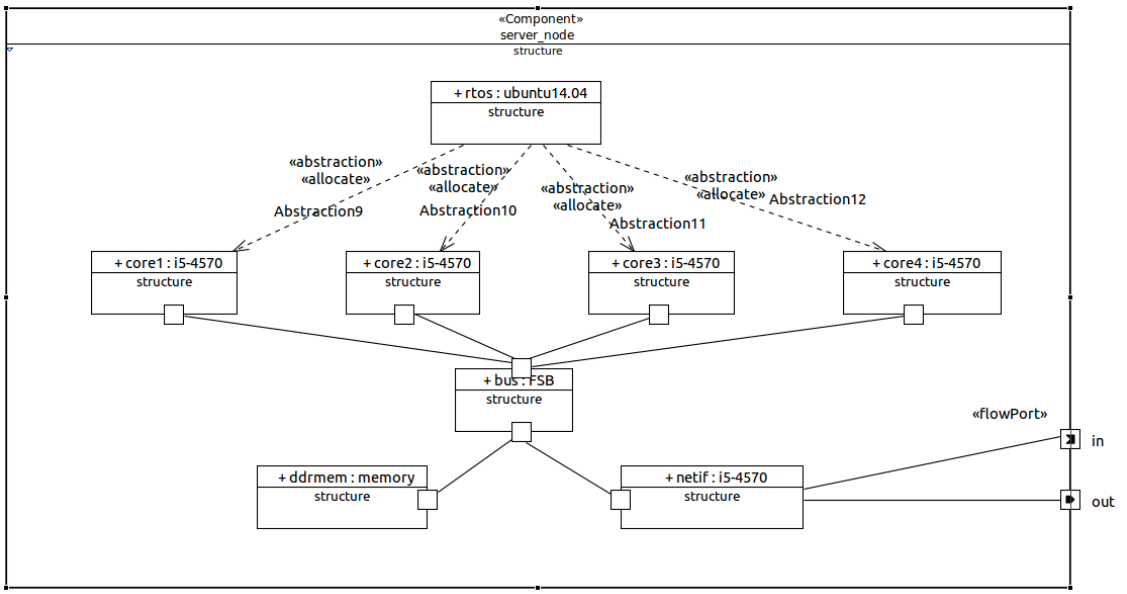


Figure 10. Example of the description of the internal architecture of a resource node.

In the example, the “rtos” UML property is an instance of the “ubuntu14.04” component type. In turn, the “ubuntu14.04” component is declared in the software platform view, as shown in Figure 11.

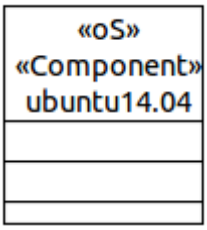


Figure 11. Declaration of an OS component in the SW platform view.

4.5.4 Description of the node as a complete system

A node can be a specific system with specific applications running on it and performing a specific functionality. Moreover, in the specific case of a distributed control system, it can perform sensing and/or actuations.

Figure 12 shows an example of node described as a complete system. The methodology supports the description of any scale of system nodes, from big servers to small embedded systems. Thus, for instance, a complete system node can consist in growing the Figure 10 SW/HW platform by including component application instances and mapping them to the RTOS instance. Figure 12 shows the case of a small embedded system, specifically, the internal architecture of the temperature sensor “T_sens_str” node, employed to model the temperature sensor nodes instanced in the Figure 4 network architecture. Figure 12 shows a node architecture where the “Tsens.exe” application (captured as an instance of a memory space in the modelling methodology) is mapped to the “rtos” instance, in turn mapped to an ARM7 based platform[5].

[5] In the methodology there is not currently bare-metal application mapping. It is assumed that every application component will be mapped to an RTOS. Eventually, the model can be simplified to map a memory space instance (“an executable”) directly to a processor, but the methodology implicitly assumes that there is an RTOS instance (of a default type) in between..

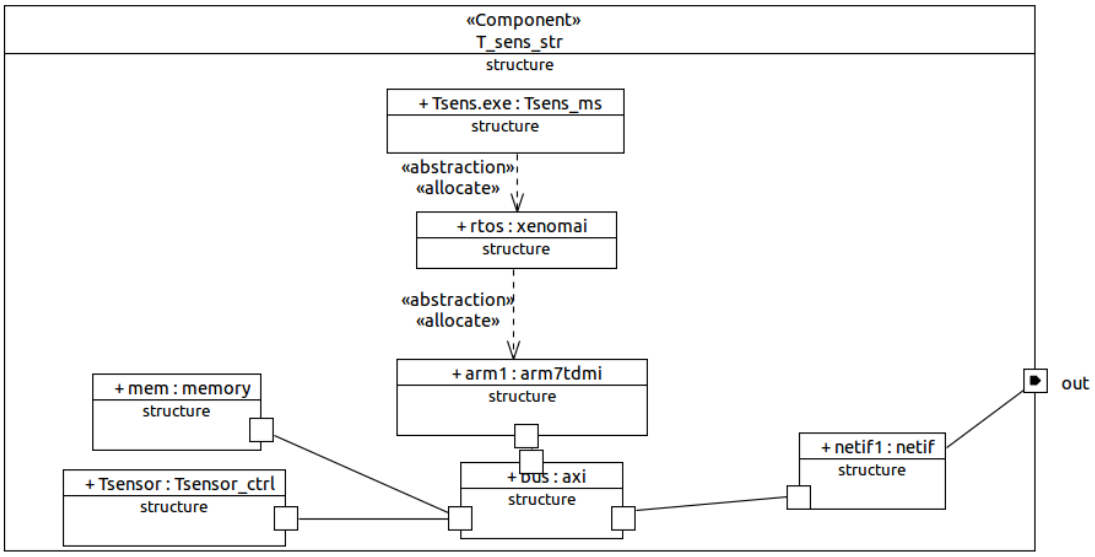


Figure 12. Internal structure of the "T_sens_str" node, example of embedded system as a node.

Remind that, in terms of SW synthesis, the memory space "Tsense.exe" corresponds to a single executable. The mapping of the application component instance to the memory space is be done in the memory space view, as reflected in Figure 13.

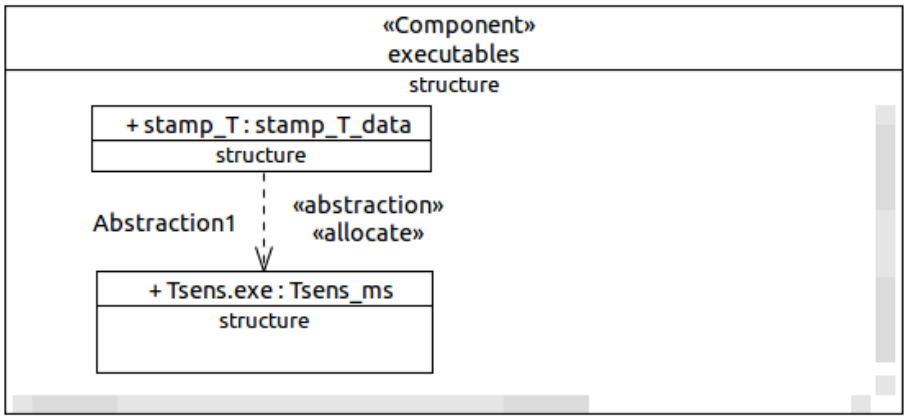


Figure 13. Allocation of a component application instance to a single memory space.

Notice that, in contrast to the single-system modelling scenario, when modelling a network node (Figure 12) "Tsense.exe" is not a reference to an existing application component instance, but it is a new application component instance. It means that each instance of the "T_sens_str" node will involve an (internal) new instance of the application and of its related memory space.

Equivalently, it is possible to capture the whole node architecture at once, at it is shown in Figure 14.

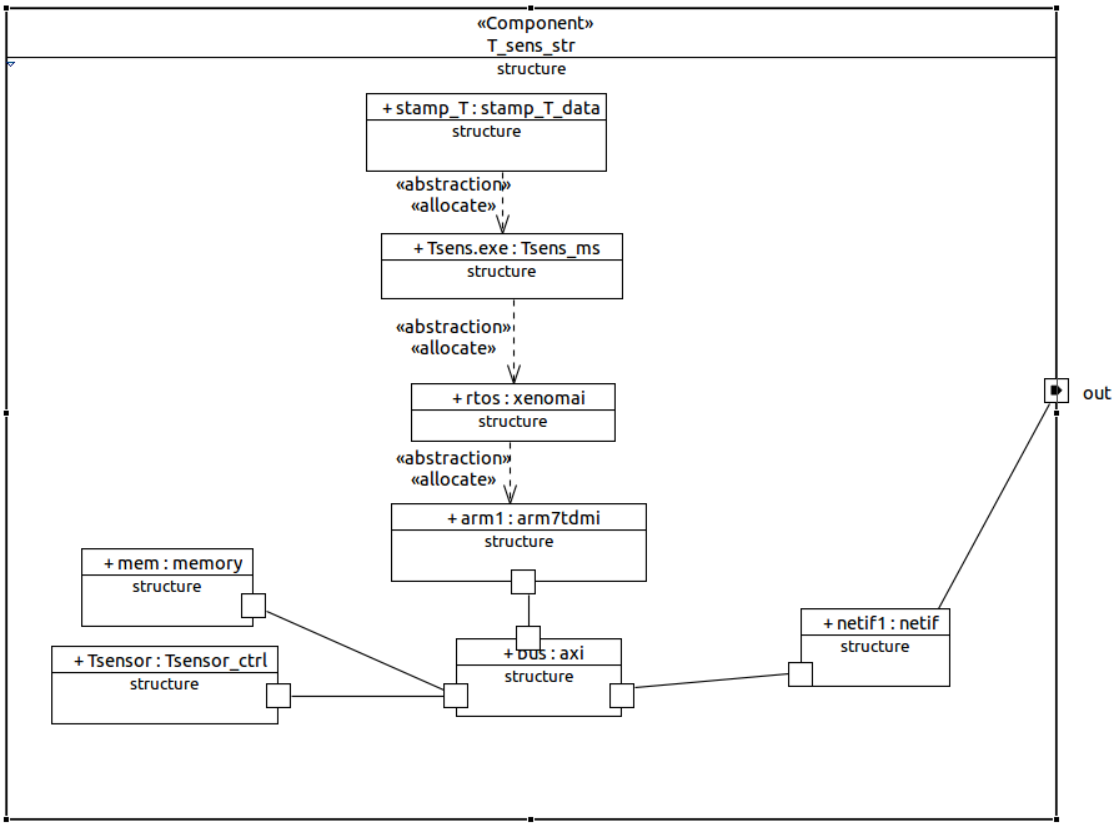


Figure 14. Allocation of a component application instance to a single memory space.

Notice again that the description of the internal structure of the node contains no references to either application instances or memory space instances from the application and memory space views respectively, but its own application component and memory space instances instead.

4.5.5 Hw Resources, SW Platform, Memory Space and Application views as Unified repositories for the Nodes´ Description

The HW resources, SW platform, Memory Space and application component views declare all the components required for every node requiring the description of their internal architecture (sections 4.5.2, 4.5.3, 4.5.4). Let's take for instance the declaration of HW resources. If two or more nodes declared in the node view are describing their internal architecture, the HW resource view has to declare all the HW resource component types employed in the description of each of those nodes. The same applies for the SW platform.

Figure 15. Example of the HW resources view. The set of declared HW components is the union of HW components instanced for each of the internal architectures captured for the three nodes declared in the node view shown in Figure 5.

4.5.6 Description of a node through a Behavioural model

A node can be described by associating a behavioural model which reflects the packet traffic generation/consumption pattern of the node. This enables an abstract modelling of complex traffic generation/consumption patterns, while it does not require to capture the details of the internal architecture of the node.

The methodology comprises to mechanisms to model the node behaviour:

- By associating a file
- By associating an activity diagram

The modelling procedure is the following. In a first step, a communicating task is declared as an application component in the application view. Such a component is stereotyped as <<Task>>, to refer that its functionality does not necessarily reflect an actual application functionality, but eventually a model of how a node will generate and/or consume packets. Then, either a file with the functionality or an activity diagram is associated to such a component.

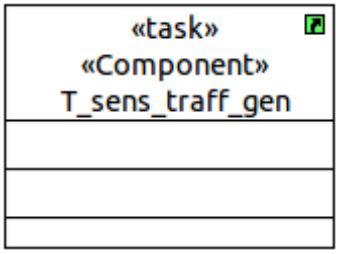


Figure 16. Behavioural model of the traffic node generation declared as <<Task>> component in the application view.

In order to associate the file, a <<file>> artifact is associated to the component. The "main" attribute of the artifact refers to the function which contains the code modelling the traffic generation/consumption.

In order to associate an activity diagram, a conventional UML modelling procedure is followed, i.e. the <<Task>> component owns an UML activity (classified Behavior) with its corresponding activity diagram. Figure 17 shows an activity diagram capturing a possible model of the packet generation performed by the temperature sensor. This way, the temperature sensor can be modelled in a more abstract way than the way it was shown in section 4.5.4. Notice in Figure 16 task component declaration a symbol denoting the association of the activity diagram[6].

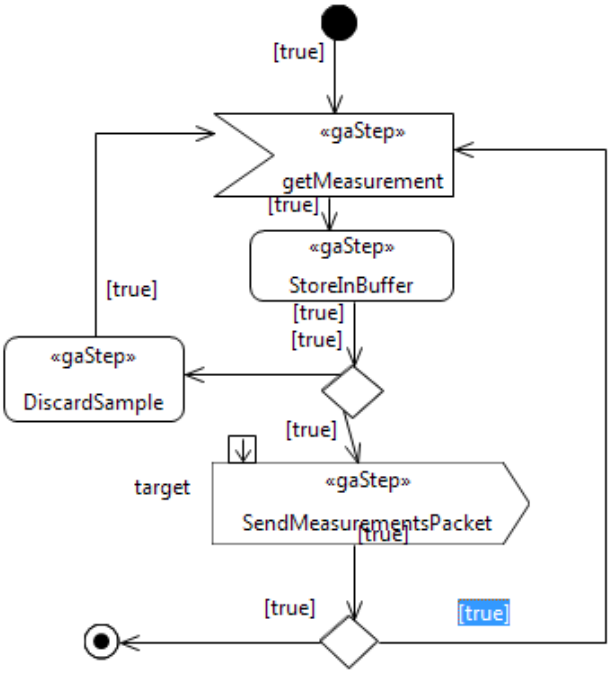


Figure 17. Example of activity diagram which captures the packet generation performed by the monolithic model of the temperature sensor node.

After the <<Task>> component has been declared and a behavioral model, either via a file or via an activity diagram, has been associated, a second modelling step has to be performed. A "void node" component has to be declared in the node view, as illustrated in Figure 18, where the "T_sens_void" void node is declared. No internal architecture description is associated to such node component[7].

[6]In the application view, this symbol also appears for the <<system>> component, which has a composite diagram associated where the application architecture is described.

[7] Notice that there is no symbol denoting that there is no association of any internal structure to the node.

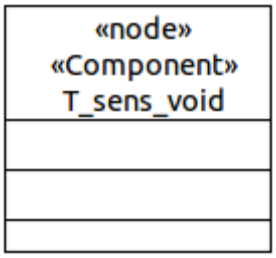


Figure 18. Declaration of "void node" in the node view.

Finally, as third modelling step has to be performed in the network view, consisting in the mapping via an <<allocate>> relationship of the task component instance to a "void" node instance.

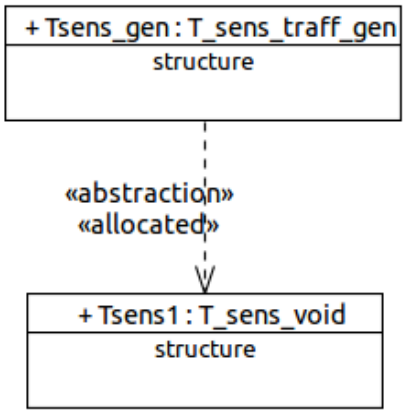


Figure 19. Mapping of a functional model of traffic generation to a "void" node.

The afore described three step mechanism is quite flexible in the sense that it allows a single-source model for the exploration of different traffic source and sink models (see section 4.9.1.2).

However, if such type of exploration is not required and the modeler is associating only a single type of behavioral model for traffic generation/consumption, then the modelling task can be greatly simplified into a more synthetic modelling mechanism. Such mechanism consists in directly associating the behavioral model to the node component (which is no longer a "void" one).

In this synthetic modelling procedure, no declaration of a <<Task>> component is required in the application view. The modeler only needs to declare the node in the node view and associate to it either a UML operation or an activity diagram.

Figure 51 illustrates a case where a "Tsensor_node" node component is declared and the same activity diagram shown in Figure 17 is associated to it[8].

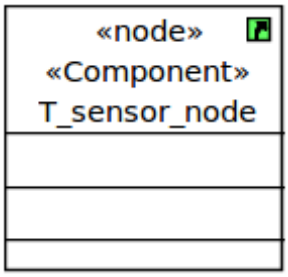


Figure 20. A component node in the node view which has been directly associated an activity diagram as behavioural model of traffic generation/consumption of the node.

Figure 21 illustrates the case where the a "Tsensor_node" node component is declared and the functionality modelling. A <<File>> artifact is used to describe the source repository of the traffic model.

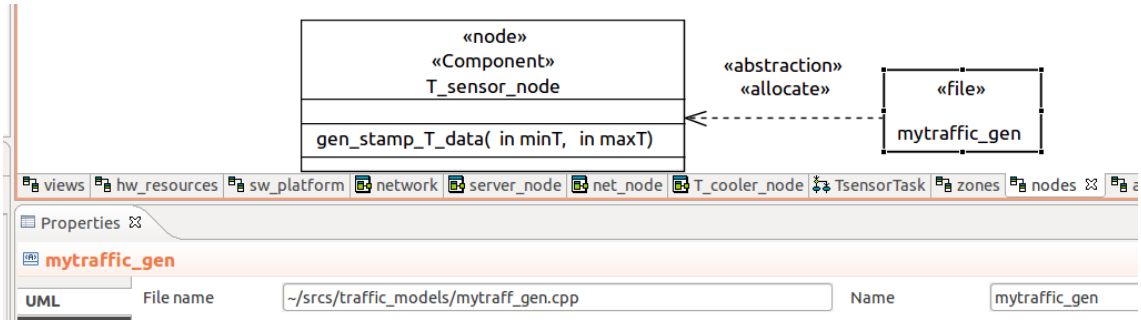


Figure 21. Direct association of a function with the behavioural model of traffic generation/consumption of the node.

Any tool navigating the model (for model transformation, code generation, etc) can recognize that the component node declared, in any of the two modelling styles, has associated a behaviour, which thus reflects a behavioural traffic packet generation/consumption model, and distinguish it from the cases where the internal architecture is described.

4.5.7 Description of a node through attributes

A node can be described through a set of attributes which define how the traffic is generated/consumed. This enables a more synthetic and abstract description than the

[8] Notice that now the node component is no longer "void", and the symbol that denotes and associated internal description (an activity diagram in this case) appears. When the internal architecture of the node is described as shown in sections 4.5.2, 4.5.3 and 4.5.4, the same symbol appears in node component declarations, but then it denoted the association of the composite diagrams employed.

one shown in section 4.5.6, for a predefined set of traffic generation/consumption patterns.

The modelling procedure consists in stereotyping the node component with the <<Node>> and the <<CommunicationEndPoint>> stereotypes.

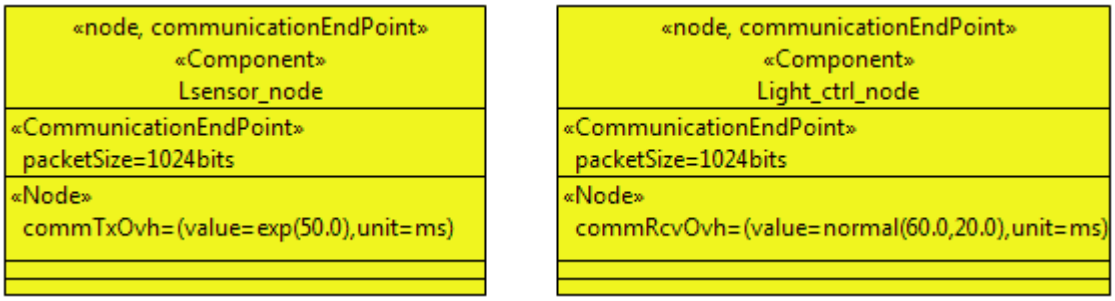


Figure 22. Declaration of two nodes which model traffic generation/consumption through attributes.

The combination of the "packetSize" attribute (provided by <<CommunicationEndPoint>>) and the "commTxOvh" (provided by <<Node>>), enable the modeling of traffic source generators. The combination of the "packetSize" and the "commRcvOvh" attributes (provided by <<Node>>) enable the modeling of traffic sink nodes.

Finally, these node components are instanced in the network view. No allocation of application or memory space instances to them is required (see Figure 23 and Figure 30) as an example.

4.6 Multi-level network modelling

The methodology supports multi-level network modelling. Specifically, it means that the network architecture enables the instantiation of nodes at different abstraction levels (see section 4.5.1).

Figure 23 shows an example of a temperature and light control system in a building, where node instances corresponding to the three abstraction levels are instanced.

Specifically, the light sensors and controllers are node instances (coloured in yellow) typed as node components specified through attributes, in order to model then as source and sink generators (see Figure 22).

The temperature sensors are node instances (coloured in pink) typed as node components with an associated behaviour (specifically using an activity diagram).

For the remaining nodes, i.e. cooler nodes (in blue), heating nodes (in orange), the four gateways and the control server (in white), the internal architecture of them is described. Figure 23 also reflects a case where all these nodes are complete systems, i.e. whose

internal description also includes the application. It reflects then a System-of-Systems as a product of the interconnection of networkable systems.

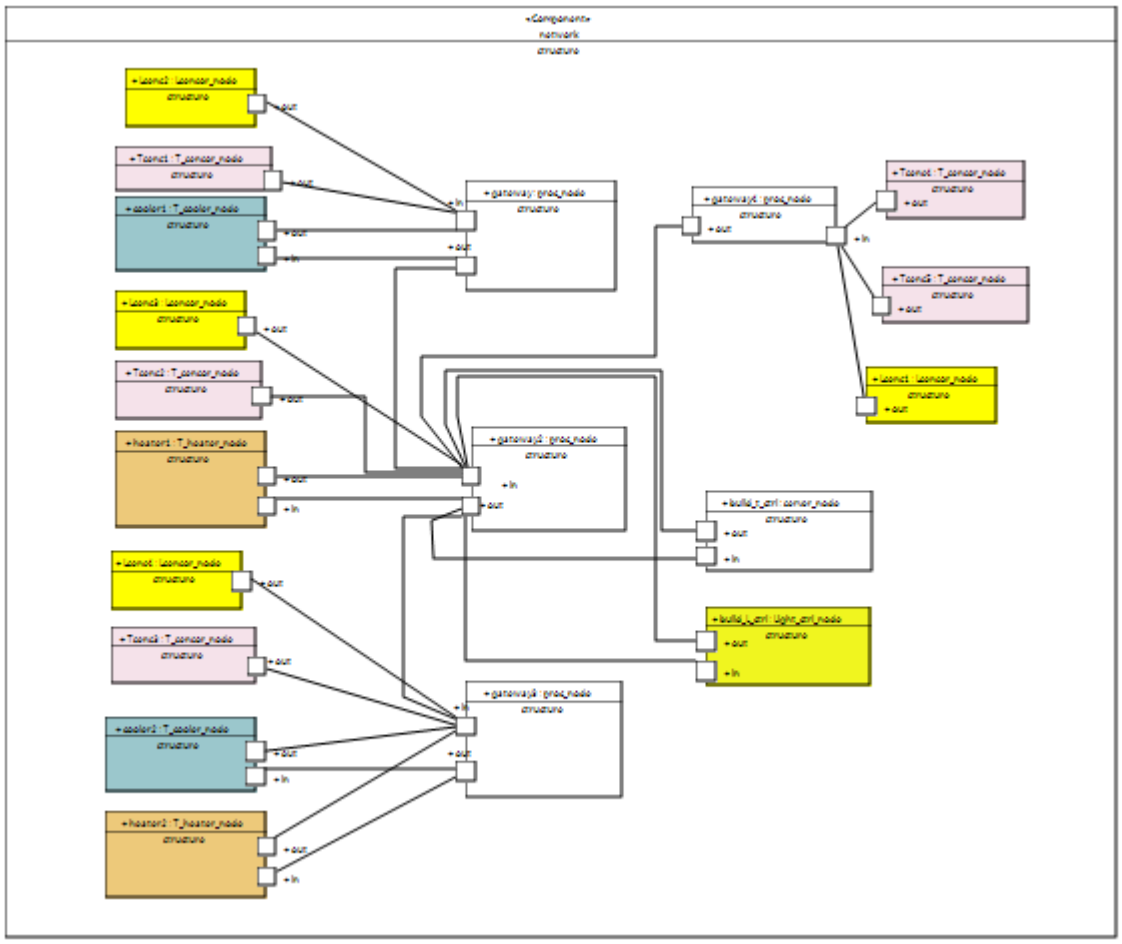


Figure 23 Example of network architecture with muli-level node modelling.

Moreover, the network model can also merge the different types of nodes explained in sections 4.5.2, 4.5.3, and 4.5.4. Figure 30 illustrates this fact. In such a model, the cooler and heater nodes are embedded systems which already integrate a specific application on top of the HW/SW platform. However, the server node is modelled as a networkable SW/HW platform (with no applications inside).

4.7 Mapping of distributed applications onto the Network

The methodology enables the description of the mapping of the components of a distributed application onto the network nodes.

Figure 30 advances that possibility showing that three component instances (in green) of a distributed application are statically mapped to the server node. Similarly, other instances (in green too) are also mapped to the gateway node. This also advances and illustrates the flexibility and power of this modelling approach. They gateway nodes are already embedded systems, with an "internal" application instance which runs a specific

routing algorithm. However, the nodes can also work as SW/HW platforms where further application components can be mapped.

In Figure 30, it has been captured in a very synthetic way. To understand its precise semantics, it is better to start first explaining how the methodology covers the mapping of a distributed application onto the network and the more explicit modelling mechanisms. Specifically, this section will cover mechanisms available to specify a single mapping.

4.7.1 Distributed Application: A PIM mapped to the network

The first thing to take into account in this methodology is that an application is always a platform independent model (PIM). What makes can make it a distributed application is that two or more component instances are mapped to at least two or more nodes of the network.

In the application view the following elements are found:

- One or more application component declaration
- one or more top application <<system>> components. Each of them has an internal architecture, with instances of the aforementioned application components.

As a general rule, the application component instances that need to be mapped to the network are only are found on such application view, within the architecture description of the <<system>> components. Each of those components reflect an application to be assessed or implemented. And each of those <<system>> applications support a broken down mapping, i.e. each internal application instance can be potentially mapped to any node of the network.

The previous rule does not mean that there are application component instances in the application view. As was explained in section 4.5.4, the description of the internal architecture of nodes can include application component instances. Therefore, this application component instances are already mapped its containing node. This static map cannot be changed from the network architecture view. It requires the edition of the internal architecture description of the nodes.

The methodology provides more flexible alternatives which enables to specify and change in a simple way in the network view the static mapping of component application instances, memory space instances and RTOS instances to the network nodes.

Such an alternative is illustrated in the following sections by means of an example. In such an example, of two applications called “top_Tctrl_app” and “DisplayBuildData”, declared in the application view, is going to be mapped onto the network “network_ex”, shown in Figure 4.

The “DisplayBuildData” application is formed by a single component. The “top_Tctrl_app” has an internal architecture, and shown in Figure 24. It contains several instances of other application components, also declared in the application view.

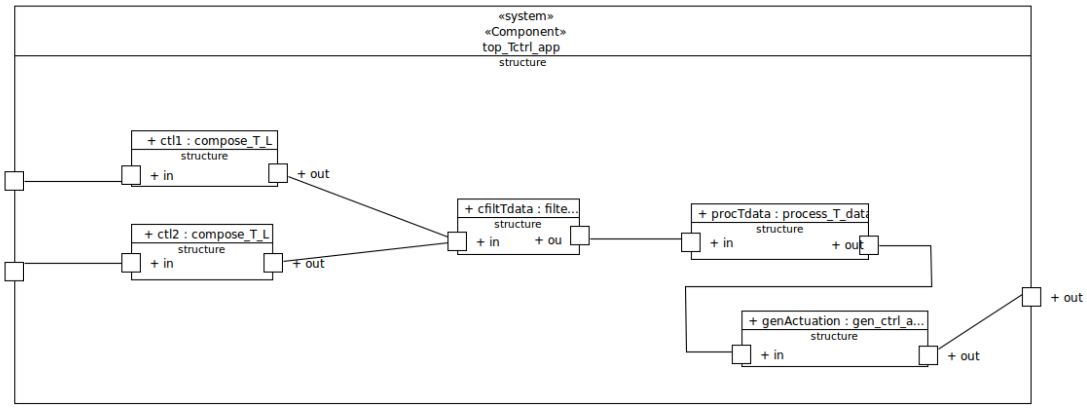


Figure 24. The architecture of the “top_Tctrl_app” application.

4.7.2 Mapping Application Component Instances onto the network

Continuing with the aforementioned example, the first step is to enable the reference to the application component instances present in the architecture of the “top_Tctrl_app” application in the description of the “network_ex” architecture. For it, in the network view, the “network_ex” component (declared in the network view) is declared as a specialization of the “top_Tctrl_app” application component (declared in the application view).

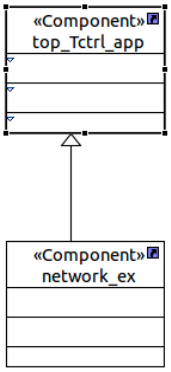


Figure 25. Generalization of the “network_ex” component to enable references to “top_Tctr_app” application component instances.

Then, in the architecture description of the network, that is, in the “network_ex” composite diagram”, the reference to the application component instances of the “top_Tctrl_app” are referenced and mapped to the network node instances. The mapping is done again via <<allocate>> relationships. This is exemplified in Figure 26 . There, “cfiltTdata”, “procTdata” and

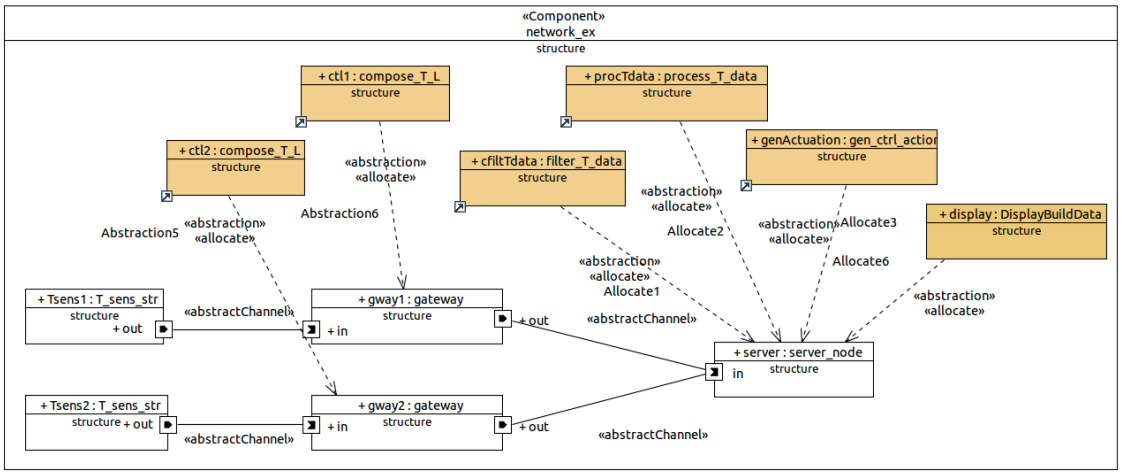


Figure 26. Allocation of the "top_Tctrl_app" application onto the network nodes of "network_ex".

The same technique used to specify static mapping of component instances into intra-node (SoC) resources is used, which helps to keep homogeneous the modelling techniques applied across MPSoC and network (SoS) levels.

Using this technique the user can simply explore the impact of different mappings of the distributed application by changing the <<allocate>> associations.

Notice also that in the example, not all the nodes have been targeted, specifically, the "Tsens1" and "Tsens2" nodes. It is not required since these nodes are already systems with its own application instance, as was shown in 4.5.4 (in charge of time stamping the temperature sample).

At the same time, the allocation of "ctl1" and "ctl2" application component instances to the "gway1" and "gway2" nodes illustrates the possibility to map application components to nodes which, in turn, already have an application component instance inside.

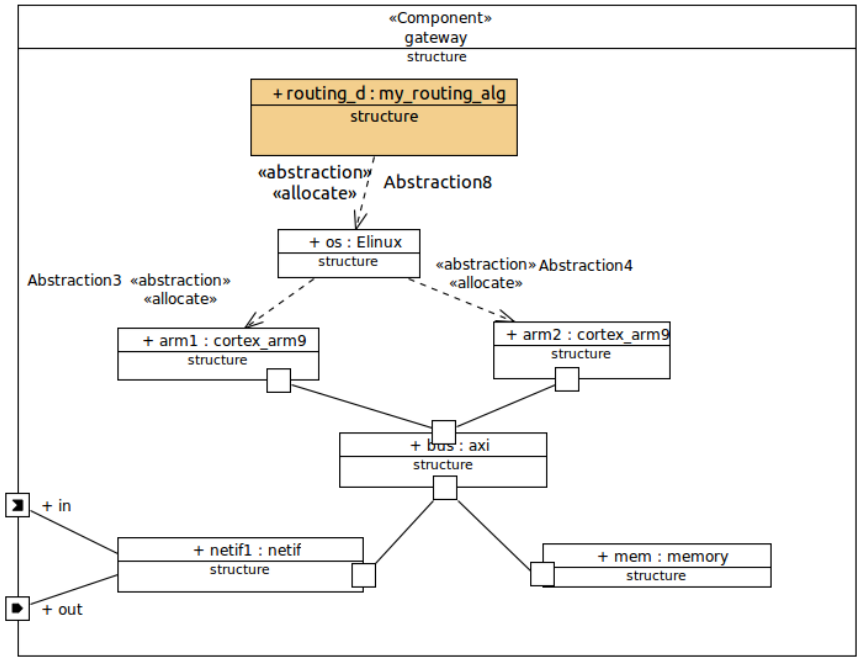


Figure 27. Internal architecture of the gateway node.

That the direct application component instance-to-node allocations shown in Figure 26, and the application component instance-to-RTOS instances shown in Figure 27 are synthetic modelling mechanisms. They are possible because they rely on a set of assumptions to enable the consideration of implicit memory space instances (executables) and RTOS instances (if required), and which enable their omission (and consequent modelling effort save).

The following rules apply for the inference of implicit memory spaces:

- Two component instances belonging to different application top components (<<system>> component) mapped to either an RTOS instance or a node, will involve two implicit memory spaces in between, one per system component.

- A direct mapping of an application component instance to an RTOS instance involves an implicit memory space in between. In other words, they become the same executable.

- If two or more application component instances belonging to the same application top component (<<system>> component) are directly mapped to an RTOS instance, it involves an implicit shared memory space in between.

- Two component instances belonging to the same application top component (<<system>> component) but directly mapped to different RTOS instances or nodes involve different implicit shared memory spaces in between.

The following rules apply for the inference of implicit RTOS instances:

- Any direct mapping of two or more explicit or implicit a memory spaces to the same processing element involves an intermediate implicit single-processor RTOS.

- Any direct mapping of either a explicit or implicit memory space to a node integrating a cluster of symmetric processing elements involves an intermediate implicit SMP-RTOS.

- Any direct mapping of a single memory space, either explicit or implicit, to a single processing element of CPU type involves an intermediate implicit RTOS[9].

- Any direct mapping of a single memory space, either explicit or implicit, to a custom processing element (FPU, etc) involves no RTOS instance.

The inference rules rely on assumptions on the mapping preferences minding performance. For instance, it is assumed that when two application component instance belonging to the same top application component are mapped to the same RTOS instance, then it is preferred to put them in the same memory space because it involves lighter and faster communications among them.

The inference rules also follow methodological criteria. The <<system>> components are distinguished from non-system components in the stronger requirement of involving at least a separated memory space.

As a result, the Figure 26 diagrams is equivalent to the diagram of

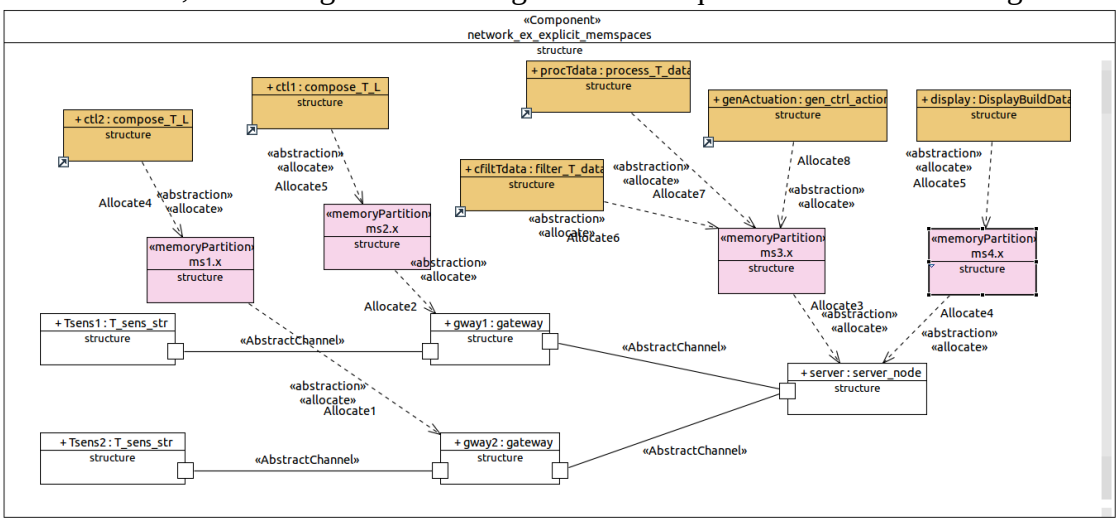


Figure 28, and the Figure 27 diagram is equivalent to the Figure 29 diagram.

[9] Modelling of bare-metal applications are not supported so far.

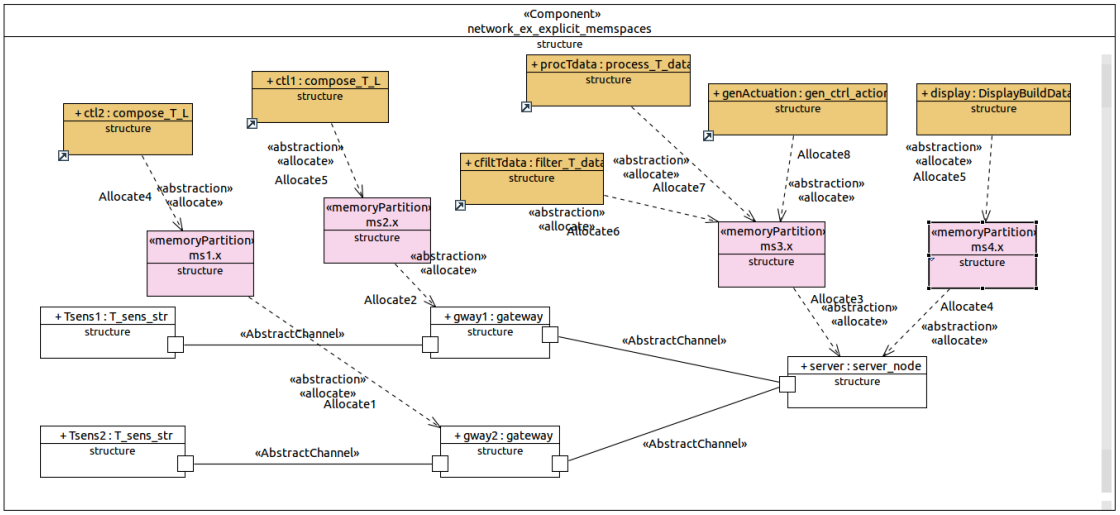


Figure 28. Figure 26 mapping making explicit memory spaces.

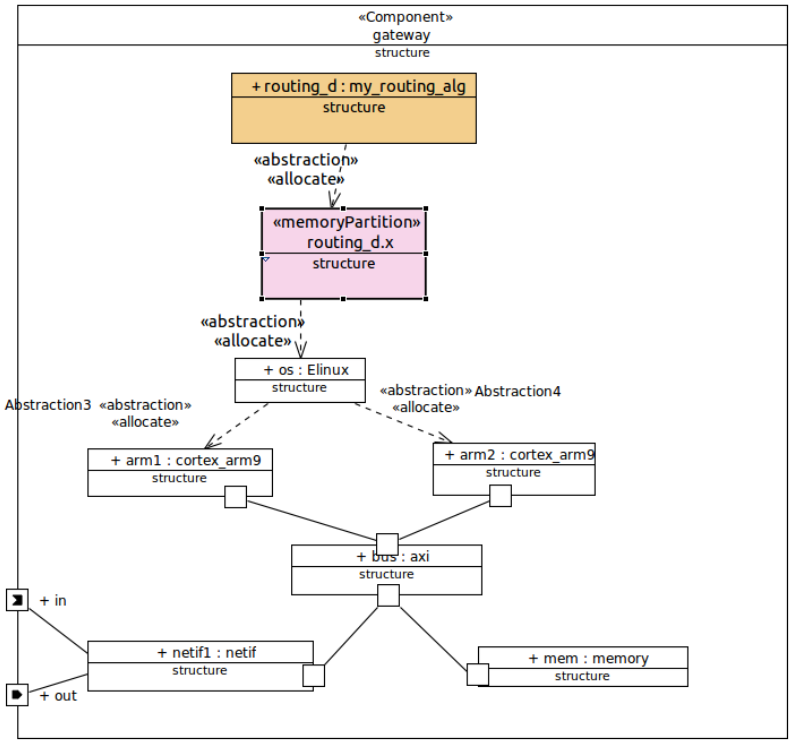


Figure 29. Figure 27 mapping making explicit memory spaces.

Finally, Figure 30 illustrates a case where the network model merges instances of nodes at different abstraction levels, and the mapping of distributed application.

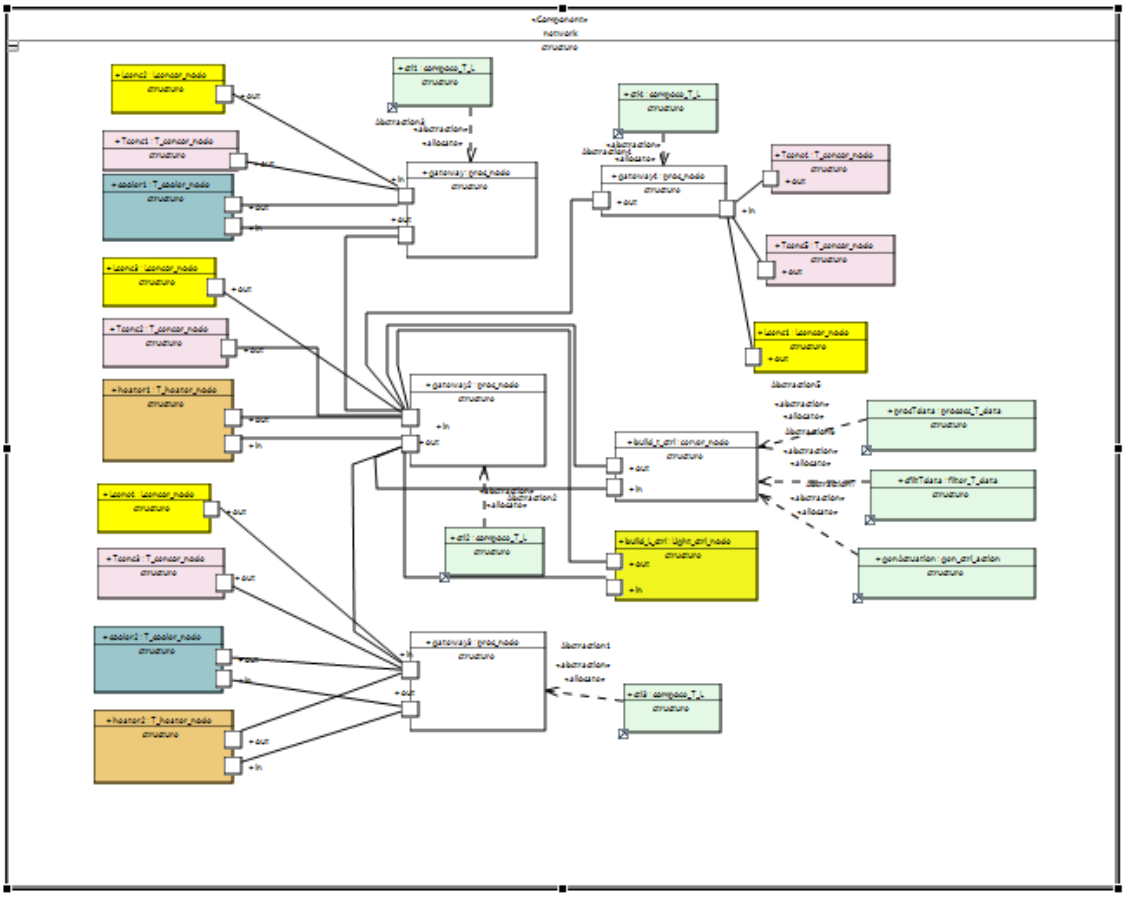


Figure 30. Example of fixed allocation of a distributed application onto a set of processing nodes of a network.

4.7.3 Mapping with explicit Memory Spaces

As was shown in the previous section, the inference rules rely on a set of criteria which enable a synthetic mechanism to specify the mapping of the distributed application onto the network, which enables the omission of memory spaces in the model, and specifically, in the network architecture model.

The production of models where the memory spaces are explicit, is not only supported, but it is necessary if the modeller wants to break the criteria of the aforementioned memory space inference rules. For instance, the modeller might want to map every single application component instance on its own memory space (as it is shown in Figure 31). In such an example, the "cfilTdata", "procTdata" and "genActuation" application component instances, which belong to the "top_Tctrl_app" are mapped to the same node (the server node), but each with its own memory space (so producing its own executable in synthesis). This also enables to explore the effect of different memory space allocations and mappings (see section 4.9.1).

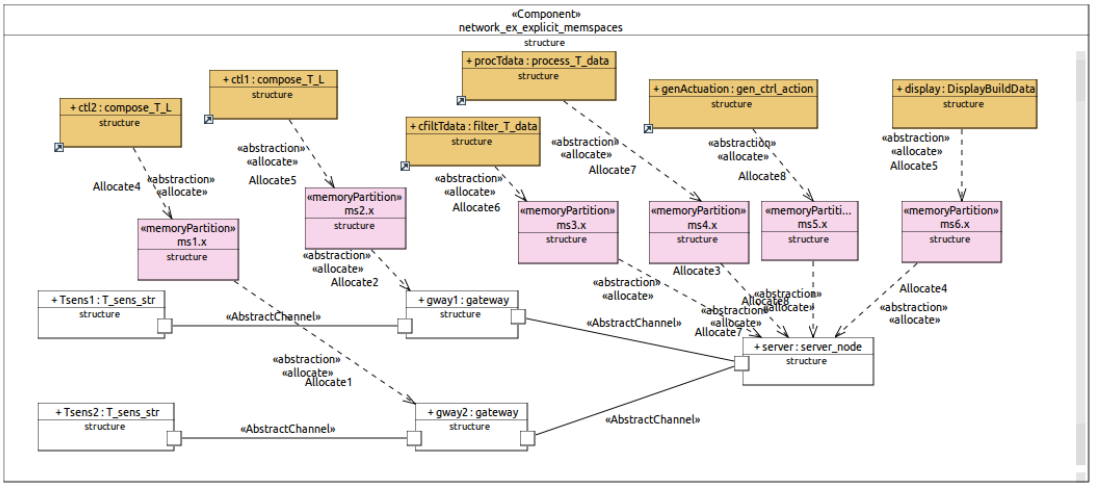


Figure 31. Mapping of the distributed application allocating specifically a memory space to each application component instance.

4.7.4 Distributed RTOS: Mapping RTOS instances onto the network

In a similar way, the methodology enables the capture of distributed RTOS. They are captured using the same technique, i.e. by means of <<allocate>> UML abstractions from the RTOS instances to the nodes. The methodology assumes that such a mapping can be done to:

- Any node which describes only a set of HW platform resources
- Any node with an internal architecture description (sections 4.5.2, 4.5.3, 4.5.4) with computational resources (CPUs) not allocated.

In both cases the default semantics is that the RTOS maps to all the available computational resources (assuming that the distributed RTOS is SMP, then the target must be also SMP).

The description of the gateway platform, shown in Figure 32, shows an example of node as a HW platform.

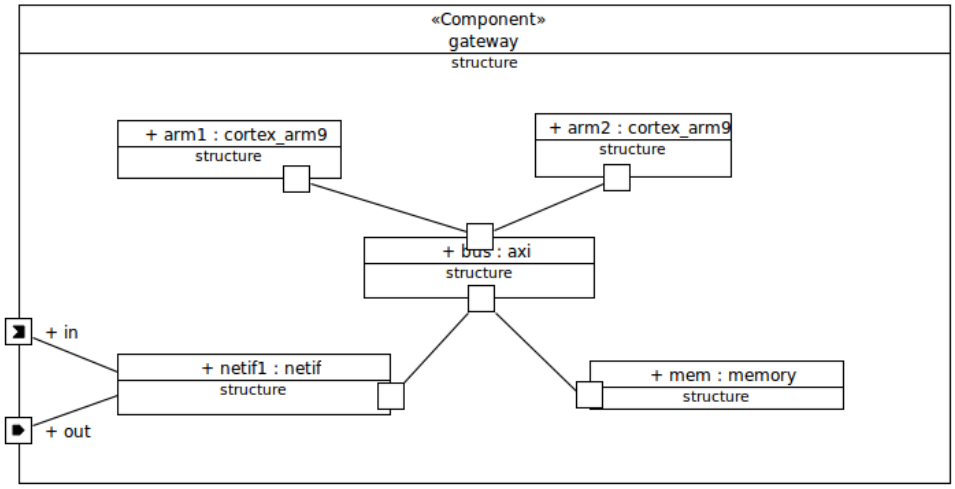


Figure 32. Description of the "gateway" node, example of node as a HW platform.

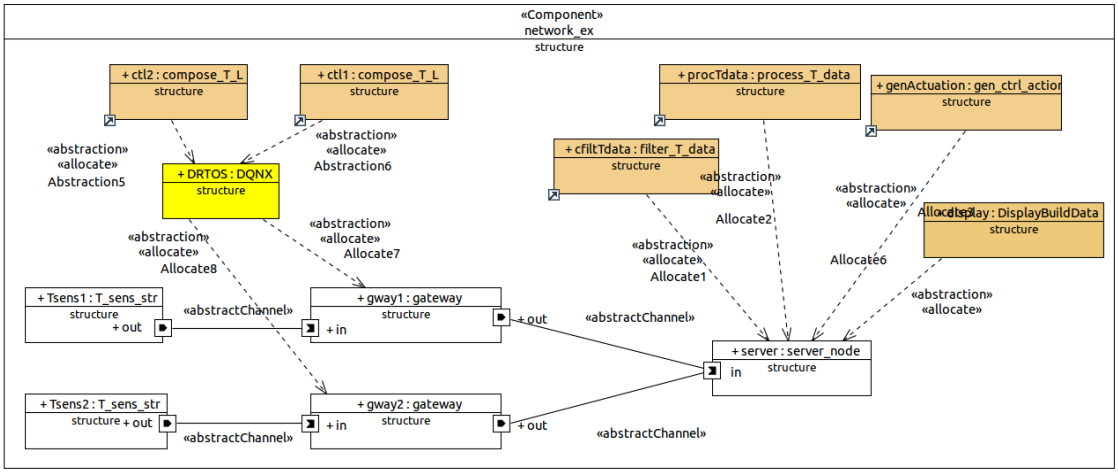


Figure 33. Example of distributed RTOS instance ("DRTOS") mapped to two nodes ("gway1" and "gway2"), modelled as two networked HW platforms.

From the aforementioned default semantics, the distributed RTOS takes over the two processors of each gateway node.

4.8 Modelling of Zones and Contiguity

The modelling methodology enables the modelling of zones in a wide sense. The modelling approach which will be described relies on a minor extension of the domain view and profile shown in section 3.1, and shown in Figure 34.

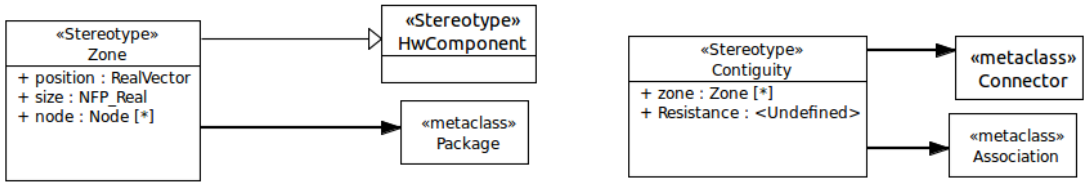


Figure 34. Zone inherits from the MARTE <<HwComponent<< stereotype (as well as being applicable to Packages, it makes Zone applicable to component and properties, among other classifiers), and Contiguity is applicable to connectors, as well as to associations.

A first modelling scenario supported is the definition of zones as areas where transmission characteristics are homogeneous, in the sense the transmission characteristics between two nodes in the same zone are the ones involved by the network architecture captured and explained in the previous sections. It obligues to consider the traversed path, the attributes of abstract channels in the path, and the traversed nodes.

Additionally, the *contiguity* modelling concept enables to consider the change or penalty in the transmission conditions when the communication between nodes has traverse zones[10].

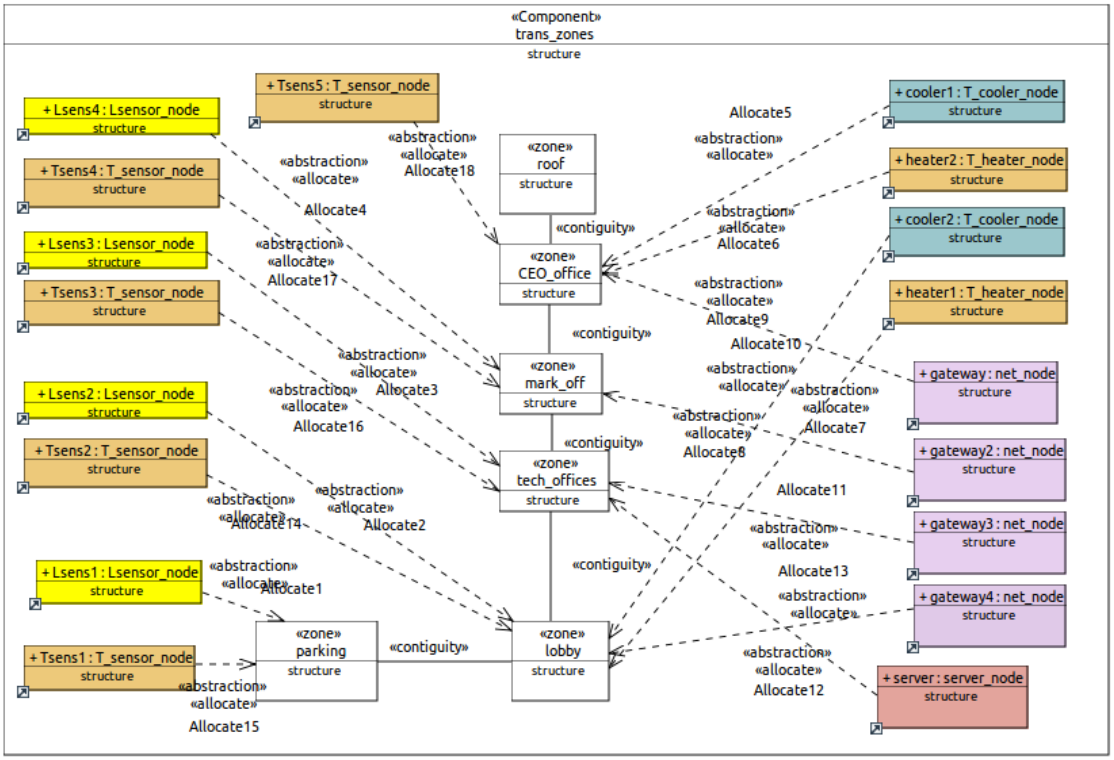


Figure 35. Example of description of zones with their contiguities.

Since <<Zone>> inherits from <<HwComponent>> (an excerpt of its definition in MARTE is shown in Figure 36), the "dimension", "area" and "position" attributes for an abstract geometric characterization of Zone are available.

[10] The basic semantics of this concepts, zones, contiguity and resistances are the same as in [1][7][8]. The change is in the modelling style which provides more flexibility and enables single-source DSE model, on the possibility to describe environment conditions, relying on MARTE HW_Component, and on the possibility to handle different zone types.

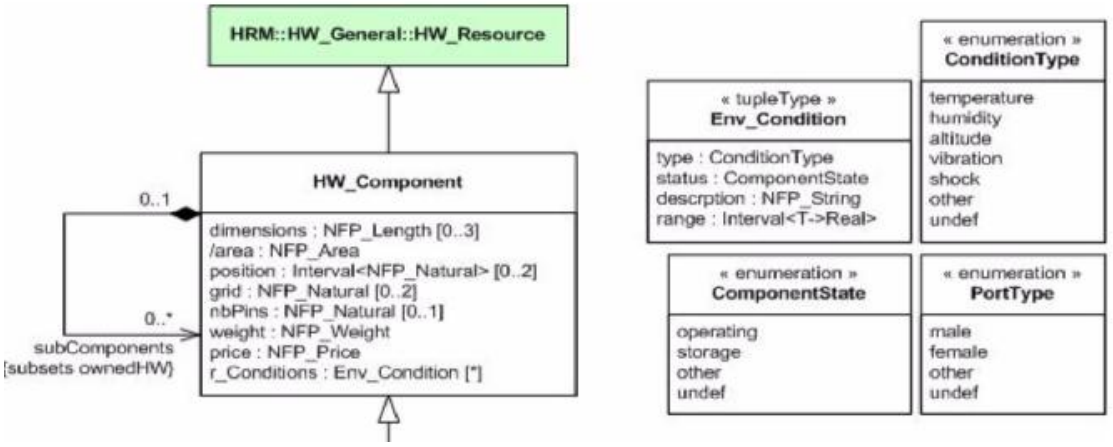


Figure 36. Excerpt of the MARTE profile where HW component is defined.

Moreover, the <<HW_component>> stereotype, and thus the <<Zone>> stereotype enable the association of environment conditions. In MARTE, they serve, in the context of <<HW_Component>>, to state the environment conditions required by the HW component.

The <<Zone>> extension in CONTREX extends the semantics to mean the environment conditions imposed by the zone.

Figure 37. Stating humidity conditions associated to the "parking" zone captured the "trans_zone" zone diagram shown in Figure 35.

Moreover, the methodology supports several zone layouts.

That is, with the previous modelling elements, one would be obliged to specify the same zone breakdown, common for every aspect affecting the nodes (propagation conditions, temperature, humidity, etc). This can be understood as a single zone layout.

By supporting several zone layouts, each layout (or zone layer) would be defined by the "type" attribute of the zone. In the Figure 37, all the zones are of "type=humidity". Let´s pose now that we want to consider the temperature conditions. In some cases, it may happen that the zones defining humidity conditions also match a layout of temperature zones. However, in many other case it might not happen. For instance, let´s assume that the parking temperature sensors have a specific temperature regulation to ensure they work over 0ºC and below 30ºC.

Then, as it is sketched in Figure 38, specific temperature zones (in this case one for outdoors, and another for "inside_Lsensor_cover") can be defined. Then the light sensors are allocated to this temperature zones. However, in terms of "humidity"

environment conditions, the sensors are allocated to the "parking", "lobby", "tech_offices" and "mark_off" zones.

Notice that for the sake of simplicity in the model "humidity" zones not reached by a light sensor also allocated to a "temperature zone" can define also a temperature condition. That is, such in such a zone there can be a match of humidity and temperature conditions. However, if a zone like for instance "parking" cannot define temperature conditions. The contrary involves that light sensor node "Lsens1" is assigned to two zones which potentially assign incoherent temperature conditions.

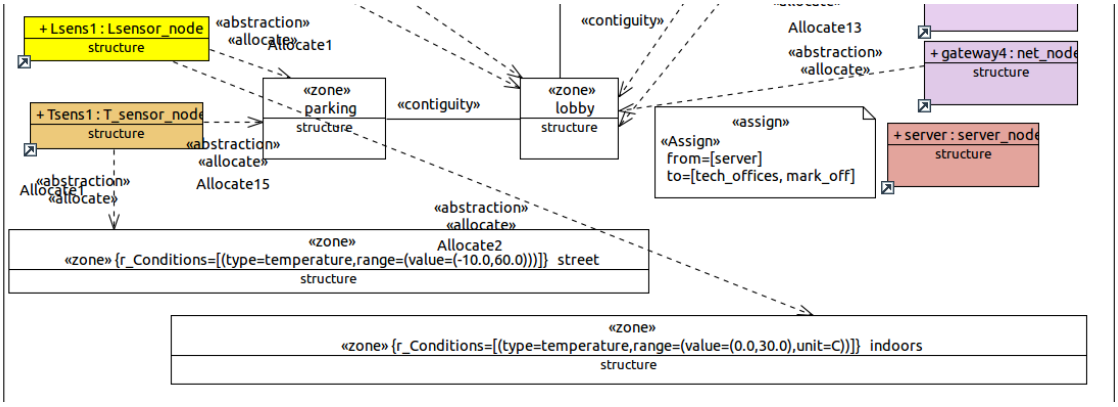


Figure 38. Different zone types are can be specified.

4.9 Single-source Network Modelling for Automated DSE

The presented network modelling methodology enables the exploration of several aspects related to the network with potential impact on the performance of either specific nodes or the whole system of systems. Specifically, it is not only possible to explore aspects which regard to the network architecture, but also aspects which refer to the networking capabilities of the nodes.

Sufficient modelling elements have been already provided to let the user edit the model for such an exploration. For instance, the capacity attributes of the network interface elements and/or of the abstract channels can be changed. The fixed allocations of application component instances or RTOS instances to nodes can be also changed. It is also possible to change the location of nodes by changing their allocation to zones (see section 4.8).

This elements provide capability for an interactive user-driven exploration. However, since it requires the edition of the model, it prevents an agile automated design space exploration (DSE). As was introduced in [3], the possibility of the description of the design space, i.e. the potential set of solutions to be explored, is required. It allows for a single-source model for DSE. It is key to avoid the time consuming editions of editing the model and regenerating the performance analysis (typically an executable) model.

This methodology enables the production of a single-source model which enables DSE of different aspects of the network model. Specifically, of:

- The attributes of the network
- The mapping of distributed application components and RTOS to nodes
- The attributes of the nodes and of their network interface capabilities.

4.9.1 Space of network attributes

It is possible to describe a design space for the exploration of different network configurations by associating sets of values to any of the attributes which describe the network.

4.9.1.1 Exploring the effect of attributes of abstract channels

It is possible to capture a single-source model for the exploration of the impact of different values on the attributes characterizing abstract channels.

Figure 39 shows a case where, over the network specification employed in previous examples, a design space of three possible values of error rates for the abstract channel connecting “Tsens2” sensor and the gateway “gway2” are illustrated.

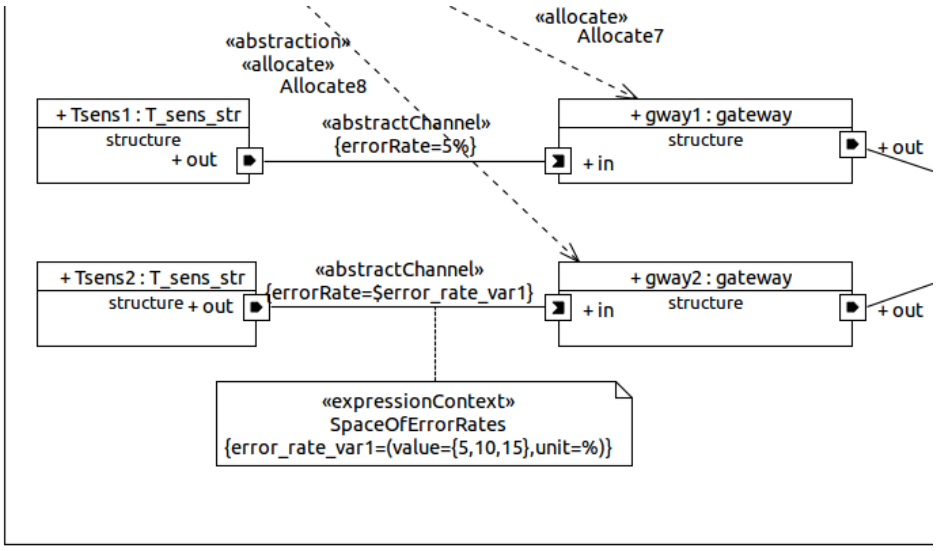


Figure 39. Specification of a design space in terms of the error rate which can be associated to one abstract channel of the network.

While the other abstract channels are configured with a fixed error rate, the abstract channel connecting “Tsens2” and “gway2” has an “errorRate” attribute with a value given by the user-defined VSL parameter “error_rate_var_1”. The conector is linked to an <<ExpressionContext>> UML constraint, where the VSL expression to define the exploration space of that variable is expressed, as stated in the CONTREX modelling methodology.

It is also possible to apply the same procedure for specifying an exploration in other attribute, e.g. the abstract channel capacity, or on other abstract channel. In the later

case, the default semantics to define the resulting design space is the cross-product. For instance, in the example of Figure 40, it turn outs into 6 combinations.

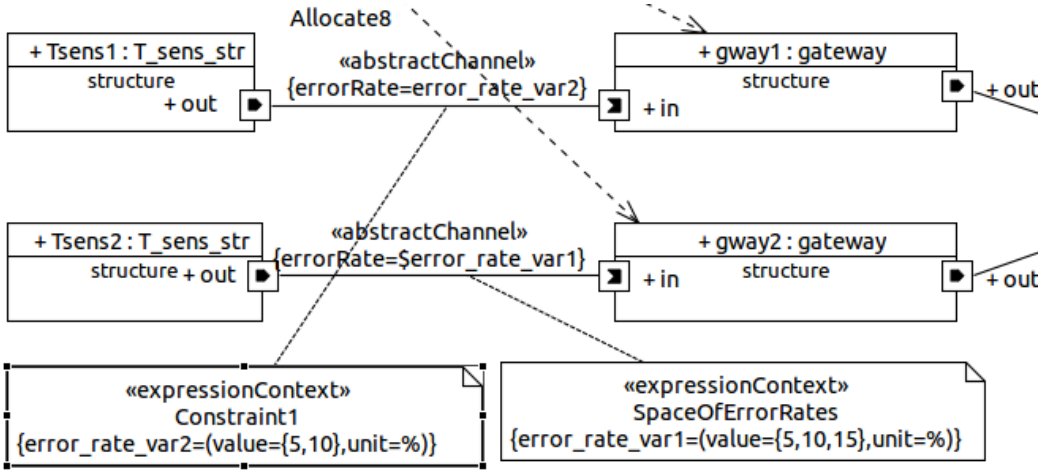


Figure 40. The exploration space grows with the cross-product of the attributes exploration space.

4.9.1.2 Exploring the effect of traffic source/sink models

The methodology supports single-source modelling for the exploration of the effect f different traffic source or sink nodes. The modelling technique is similar to the one shown in previous section.

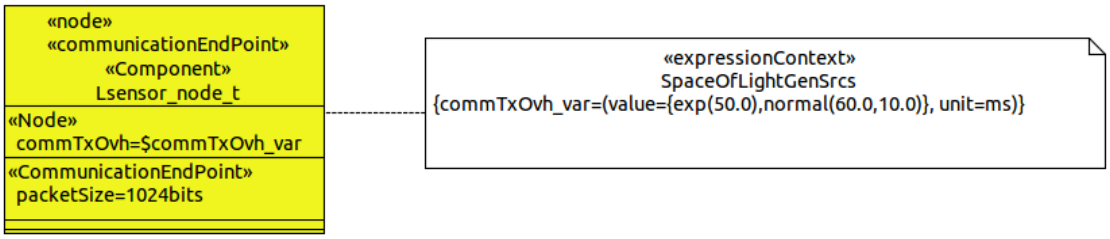


Figure 41. Stating two different traffic source distributions in a single model.

Figure 41 illustrates the case of a node which models a light sensor node through attributes (section 4.5.7) is declared. Whole the packet size attribute is fixed, the value of the communication overhead is annotated with a VSL variable ("commTxOvh_var"). Moreover, the node component is annotated with a <<ExpressionContext>> comment which states the values which the variable can adopt in the exploration, i.e. exp(50.0) ms and normal(60.0,10.0) ms. This way, in the former case the node behaves as a generator of 1024 bits packet every 50.0ms on average, with an exponential distribution function. And in the latter case, the node behaves as a generator of 1024 bits packets every 60.0ms on average and 10.0ms of standard deviation, with a Gaussian distribution function.

In the Figure 41, the comment is associated to the component node declaration. Again, if the exploration of the node attributes is to be applied to a specific instances or set of instances, it is sufficient to associate the comment to the node instance in the network view.

4.9.1.3 Exploring the effect of attributes on nodes whose internal architecture has been described in detail

The same techniques applied for single-source modelling for DSE explained for single-system modelling apply. They include exploring attributes of components, and of mappings at the different levels of the model.

4.9.2 Space of network mappings

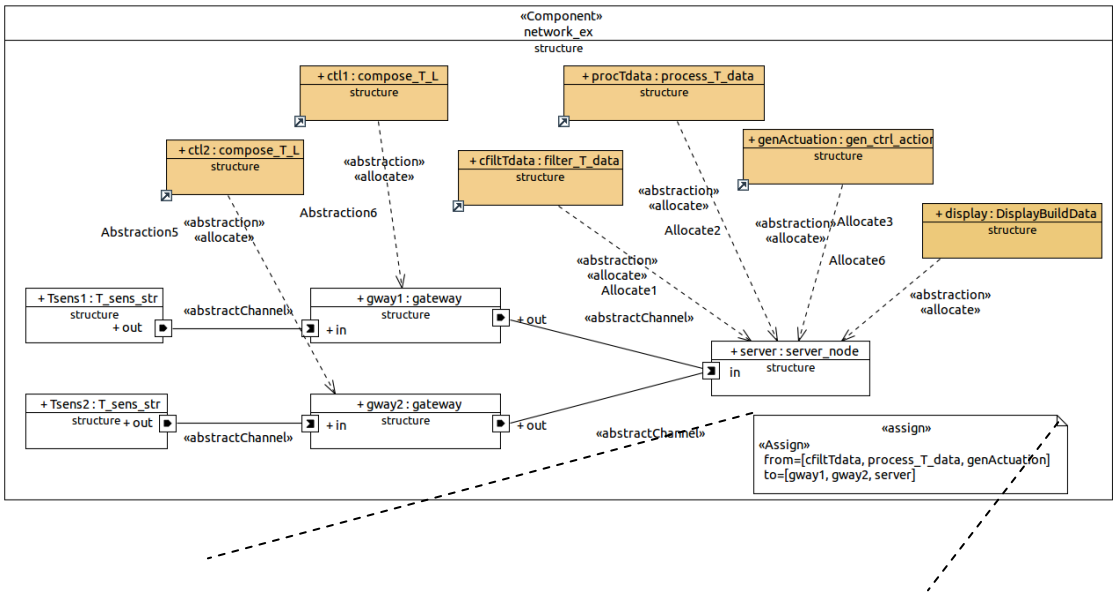
4.9.2.1 Exploring direct mappings from the application into the network

The methodology enables a single-source model for DSE of then of mappings of the distributed application onto the network architecture.

For it, the <<assign>> stereotype is used. Figure 42 shows an example of description of a set of mappings from a set of application components instances ("cfiltdata", "procTdata" and "genActuation") directly onto a set of network nodes ("gway1", "gway2", "server"). It is a variation of the Figure 26, which keeps the mapping of "ctl1" and "ctrl2" application component instances fixed, while it states, through the <<assign>> stereotype that the application component instances can be mapped either to the "net_node1" node, to the "net_node2" node or to the "server" node. Since in this case, memory spaces are omitted (for the purposes of the example let´s suppose that the gateways are complete systems), the inference rules introduced in section 4.7.2 apply.

The semantics of the <<assign>> is that it defines all the variations of the n-th "to" elements (nodes) taken in m-th elements of "from" (component application instances in this case). The result is in this case m^n (3^3=27) variations.

Notice in the example that the <<assign>> comment element can be in the model with the <<allocate>> fixed associations. When the model is used for exploration, the <<assign>> overrides the fixed mapping, while the fixed allocation server as default mapping, which can be used to produce a fixed performance model or for synthesis.



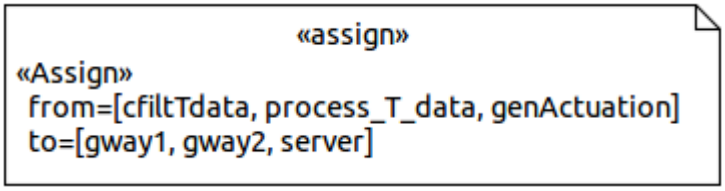


Figure 42. Example of capture of a set of mappings of the "Top_Tctr_app" application on the "network_ex" architecture.

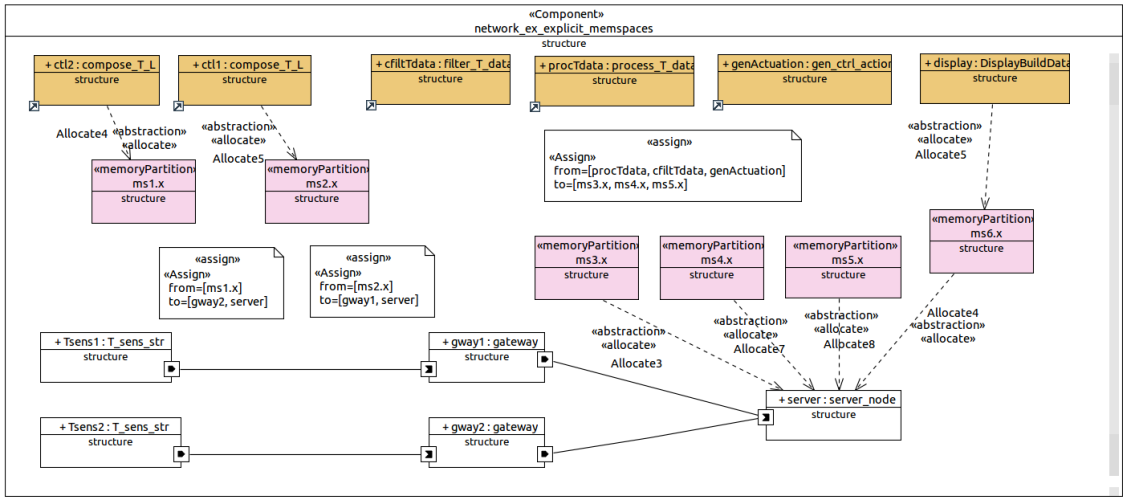
As has been shown, the application to network mapping enables to keep a homogeneous description style[11].

Similarly as in the single system description case, constraints to reduce the exploration space can be applied, for instance to eliminate exploration of symmetric solutions.

4.9.2.2 Exploring Mappings at different levels

A single-source model for the DSE of memory spaces handling. In the example of Figure 43, the modelling of design space specification for two types of mappings is shown.

Figure 43 shows the specification of a design space considering the possible mapping of three component application instances ("cfilTdata","procTdata", and "genActuation") into three different memory spaces ("ms3.x", "ms4.x", and "ms5.x") is modelled. This enables, for instance, a single-source exploration of the impact of mapping to different memory spaces, and study if the different communication overhead is assumable if the user is interested in separating the components into different executables.



[11] This is an adaptation of the modeling technique used to specify mappings of component applications to platform resources at an intra-node level. The main change is that the destination of the <<allocate>> association is a node instance (not a processing resource of the node).

Figure 43. Example of exploration of mappings of component applications to memory spaces and memory spaces to nodes.

Figure 43 also illustrates the specification of a space of mappings of a memory spaces to nodes. Specifically, the model states that the "ms1.x" memory space can be mapped, so executed either on node "gway2" or on node "server". Similarly, the model states that the the "ms2.x" memory space can be mapped either on node "gway1" or in node "server".

Therefore, Figure 43 defines up to $2 \cdot 2 \cdot 27 = 108$ mapping configurations[12].

4.9.2.3 Mapping of distributed RTOS to nodes

In general it should be sufficient to specify which nodes work under the management of a distributed RTOS.

However, the use of the <<assign>> will help to explore the allocation of component instances or memory spaces to different distributed RTOS instances or to an RTOS instance or a node.

4.9.3 Space of Node Interface Capacities

As was mentioned in section 4.5.2, the type of network interface device available and used by a node to connect to the network may have a relevant impact on the performance of both the node, and the overall system-of-systems.

For instance, it is possible to have an over-dimensioned amount of network resources, of servers providing a high computing capability, and of sensor nodes work at suitable speed. However, if those sensor nodes have, for example, a too slow network interface to transfer the sensed data, then overall performance of the control distributed system ca be unsatisfactory.

Because of that it is important to enable the exploration of the impact of network interface capabilities.

4.9.3.1 Exploring the attribute values of a generic network interface

The first approach available in the methodology is to declare the attribute value as a variable in the network interface component declaration in the hardware resources view (see an example in Figure 44).

[12] In this example is seen again the interest of constraints on mapping to eliminate symettries. For instance, in the example, a mapping of the three application component to either "ms3.x", "ms4.x" or "ms5.x" can be considered equivalent.

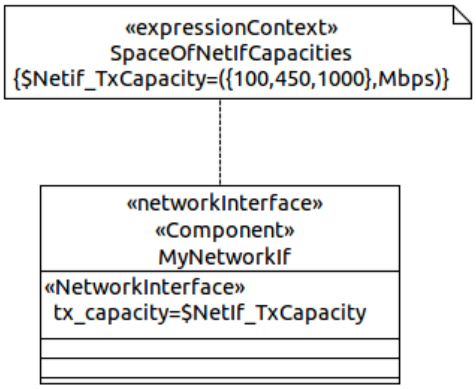


Figure 44. The transmission capacity of the "MyNetworkIf" network interface is declared as a variable. The design space is growth by stating through a constraint associated to the network interface component that three values for the transmission capacity will be explored.

This way, the design space can be described, as explained in D2.2.1 [1], by means of a VSL expression in a UML constraint with the <<ExpressionContext>> stereotype applied and associated to the component. For instance, the Figure 44 example, states a design space where the effect of using network interfaces with either 100Mbps, 450Mbps, or 1000Mbps can be explored.

Figure 44 example refers to an exploration on the value of an attribute of the component, as it is declared in the hardware resources view. This means that the value stated for the attribute applies for all the network interface instances.

The methodology also enables to state the exploration at instance level. For it, the constraint describing the exploration space is applied to the instance, within the internal architecture description of the node (composite diagram of the node within the node view), as it is shown in Figure 45.

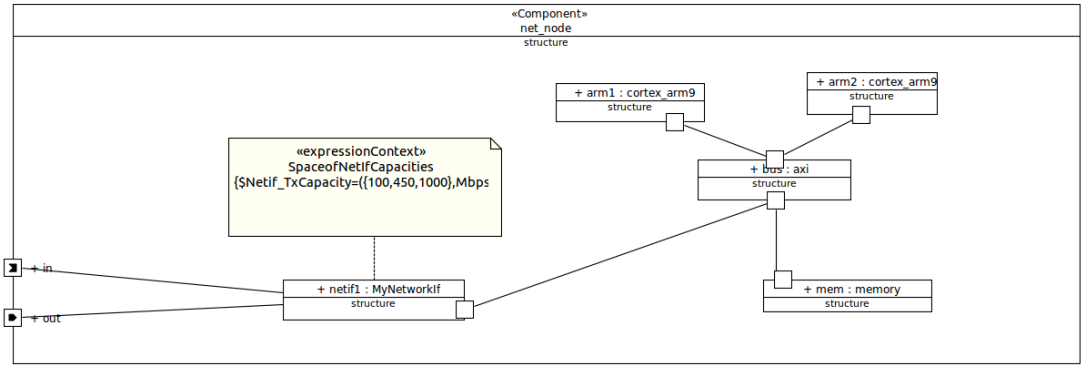


Figure 45. Declaring the space of values for the transmission capacity of a specific instance "netif1" of a network interface component.

4.9.3.2 Exploring the utilization of different types of network interfaces

A common exploration scenario which is also supported by the methodology is when the user wants to explore the utilization of different types of network interfaces. Each interface has associated a specific set of values for the network interface attributes.

The modelling technique employed relies on the definition of as many refinements of a generic network interface within the hardware resource view, as specific network interface elements one want to explore. This is illustrated in Figure 46[13].

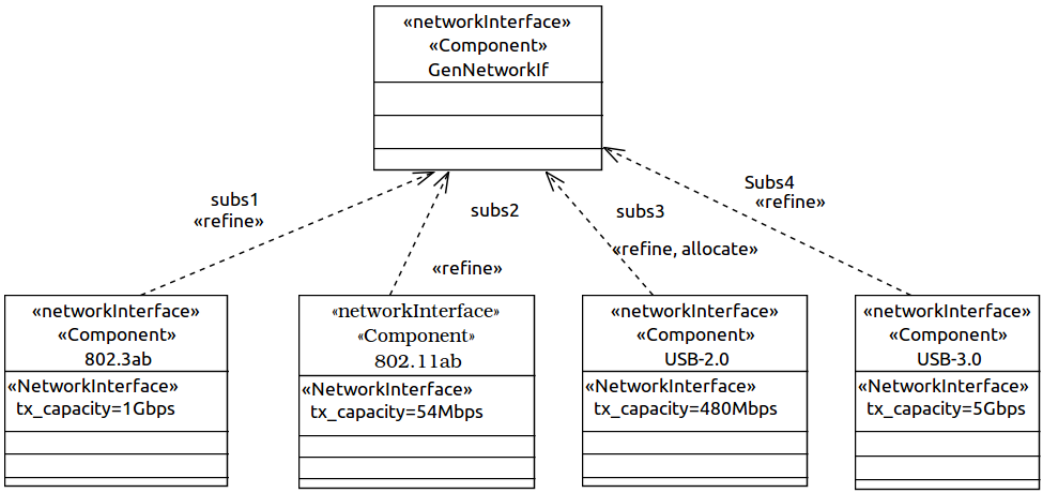


Figure 46 The network interface types to be explored are declared as refinements of a generic network interface.

The refinements are again components with the <<NetworkInterface>> stereotype. Each component states the specific set of values associated to such network interface. They are defined as refinements of the generic network interface component (named "GenNetworkIf" in Figure 46 example) by means of UML substitution associations with <<refine>> stereotype.

In addition, the <<allocate>> stereotype can be used for one of those refinement associations. It will serve to state the default type to be employed (if not exploration is performed with the model), and for software synthesis purposes.

When the model is employed for exploration purposes, by default all the refinements will be considered. In Figure 46 example, the four alternatives, namely 802.3ab, 802.11ab, USB-2.0 and USB-3.0 would be explored.

Additionally, the user can constrain the space of network interface types by means of a <<ExpressionContext>> constraint. It is illustrated in Figure 47, where it is specified to explore only the wired interfaces (802.3ab, USB-2.0 and USB-3.0).

[13] An immediate approach could be just to change the type of the network interface instance. However, it requires to change the model and prevents the single-source approach.

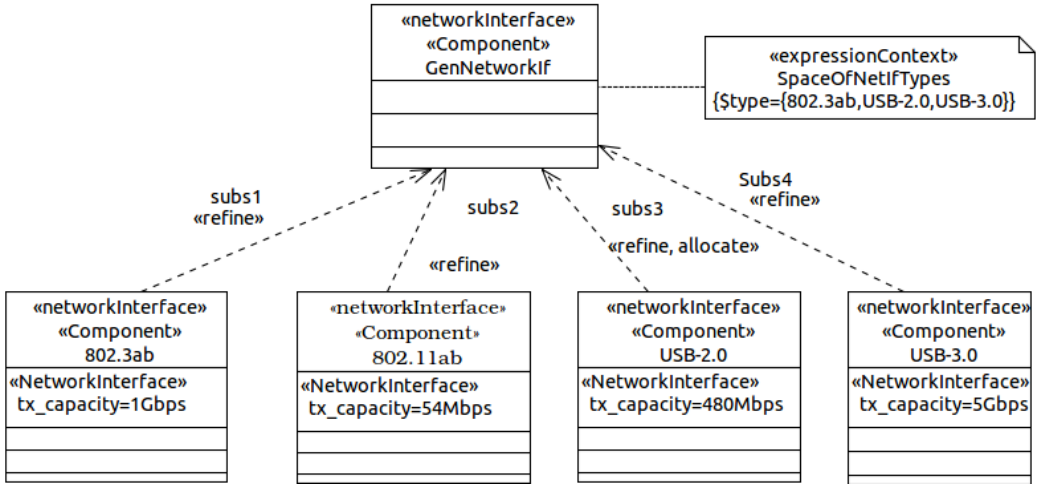


Figure 47. Constraining the space of network interface types to be explored in the hardware resources view.

The constraint of Figure 47 reflects the case where the design space is constrained at component declaration time (in the hardware resource view). It is also possible to constraint the space of network interface type exploration at instance level. It is illustrated in Figure 48.

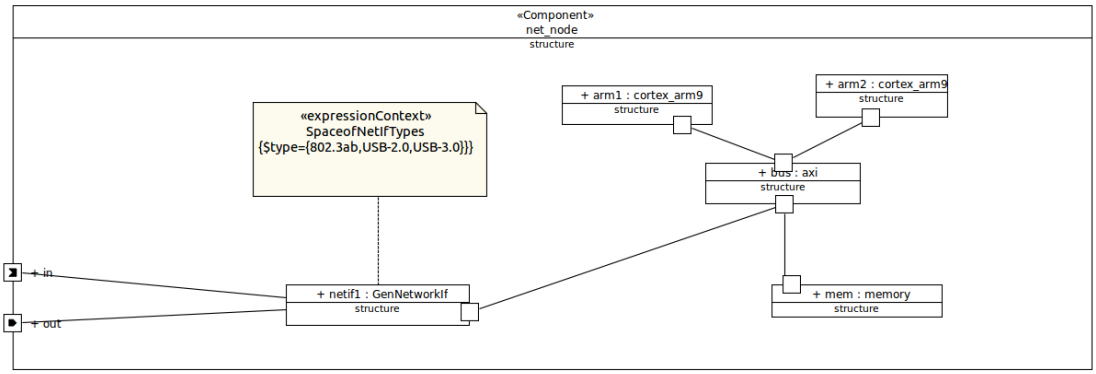


Figure 48. Constraining the space of network interface types to be explored at instance level.

4.9.4 Space of Node Distributions and Zone conditions

The modelling methodology supports the specification of different node distributions and node conditions in the model for DSE.

By reusing the modelling mechanisms explained in the previous sections, different allocation of nodes to zones can be described in a single model. Consider the example of Figure 49. In such an example, an <<assign>> comment is employed to specify that the "server" node can be located either in the "tech_offices" zone or in the "mark_off" zone.

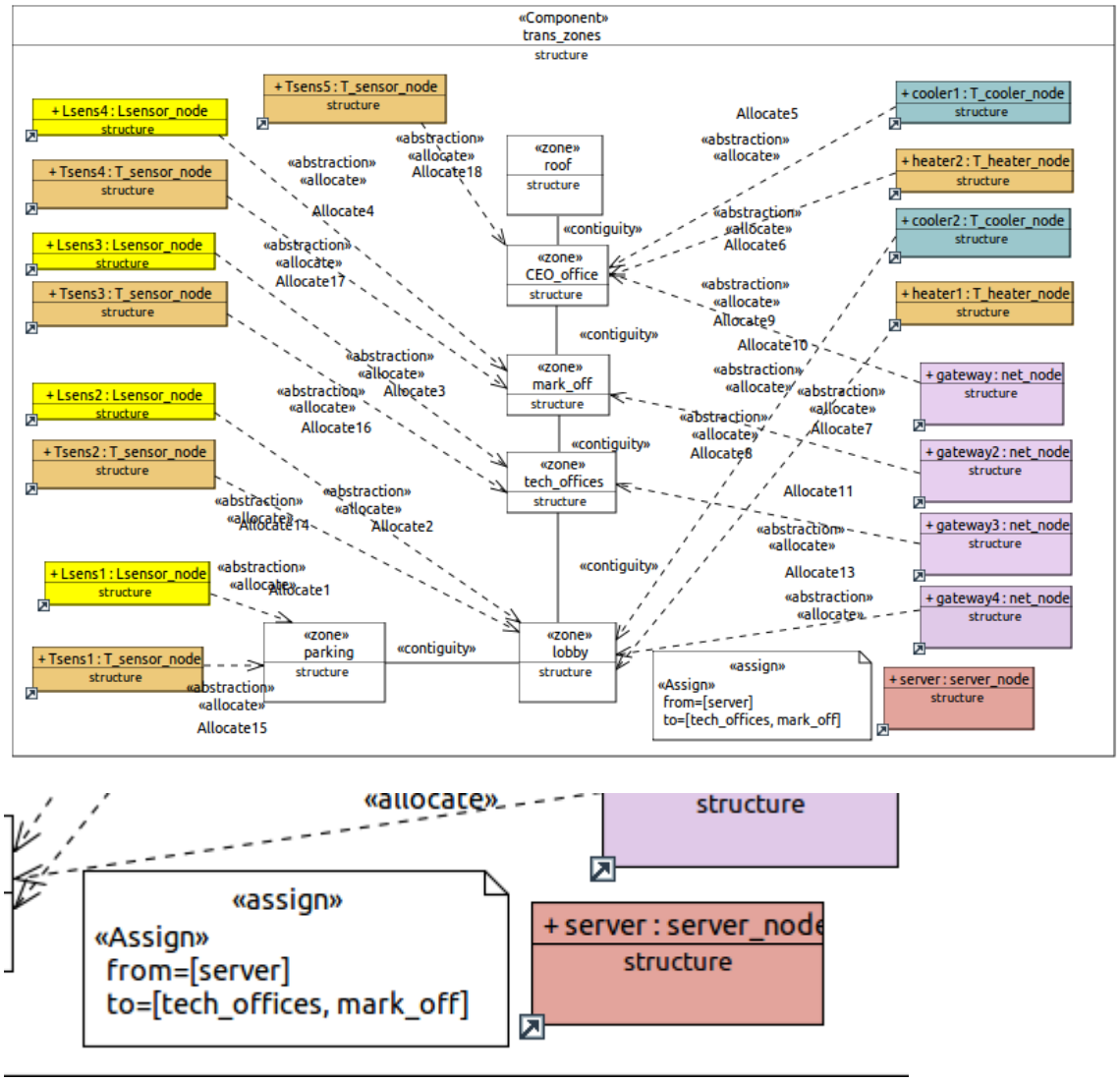


Figure 49. Example defining the placements of the server node in two zones.

Another aspect supporting single-source modelling for DSE is the specification of different node conditions.

Figure 50. In general, the specification of the environment conditions is a range.

Since the specification of the environment conditions "r_Conditions" attribute, involves the specification of a range, this can be already interpreted at DSE time, as the range of values to stimulate the performance model during the exploration. Additionally, an <<ExpressionContext>> comment can be used to express a different range.

4.10 Modelling of control distributed systems and cyber-physical systems

The CONTREX modelling methodology enables the modelling of a single system or of a networked system-of-systems. Specifically, the focus is the modelling of mixed-criticality embedded systems and mixed-criticality distributed systems for control. These systems or systems-of-systems typically sample an act on a physical environment. This environment can be modelled as a set of differential equations or similar approaches (Z-transform, state-equations, non-linear elements, etc). The connection of this type of environment models with the system or system-of-systems modelled in CONTREX build up a cyber-physical system (CPS) or system-of-systems (CPSoS) model.

The link between the embedded system or the embedded distributed system with the physical environment is two fold. There is a logical connection in the logicap ports of the application components. These ports are associated to the instances of <<HwSensor>> and <<HwActuator>> controllers, which can be either in the architectural view of the <<system>> component, but also in the node architecture description[14].

[14] Here, the frequency parameter could be used, but a better approach is to add a capacity attribute to the HWSensor and HWActuator.

5 Notes on the Compatibility with Edalab&U.Verona methodology

The shown methodology relies on a slightly extended metamodel which therefore facilitates compatibility with Edalab&U.Verona.

Following, some additional clarifications are provided.

5.1 Declaration of nodes

In the node view, abstract nodes could also have the <<Task>> stereotype for compatibility purposes- This way, it would be explicitly stated that the node has associated an abstract model (either through a behavior or a set of attributes) of traffic source/generation. It will also enable node links through dataflows (see next section).

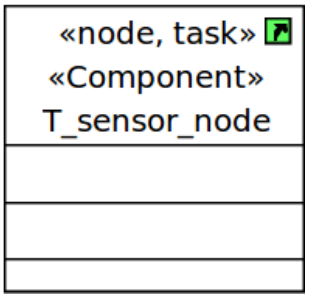


Figure 51. Direct association of a behavioural model of traffic generation/consumption to the node.

5.2 Modelling of dataflows among abstract nodes

In section 4.5.7, the modelling of dataflows to complement the modeling of traffic source and sink generators is supported by means of communication diagrams.

To rely on <<Dataflow>> stereotype is possible for compatibility purposes. As shown in the previous section, the task stereotype can be applied to the node component. The <<dataflow>> can stereotype the messages in the communication diagram shown in section 9.1.

An additional modelling style is to use properties stereotyped with <<Dataflow>>, used to link node instances. They have to be captured in the network composite diagrams as instances (i.e. UML properties) with the <<Dataflow>> stereotype applied. Such dataflow instances can then link all the task stereotyped nodes (or instances of task stereotyped components).

6 Implementation Notes

A number of implementation issues have been identified. These issues condition the UML/MARTE and model-to-text toolchain version selected, and the modelling technique and profile solution.

6.1 Redefinition of CONTREX profile prevented when referring MARTE NFP sub-profile elements

The MARTE profile source is not available and the extension is being performed through an additional profile.

However, in the later MARTE release (v1.0.1) the profile cannot be defined when an element of the profile inherits an element of the MARTE NFP sub-profile. The definition is required to edit and complete the profile, and to later be applied and used in the models.

The following solutions were tried without success:

- Eclipse Luna and older versions (June included) and their related Papyrus versions
- The installation of an older version of Papyrus on Eclipse Luna

The following know solution was verified:

- Using a Eclipse Indigo and its related Papyrus 0.8.2

The main problem is that this is a relatively old environment. It is worrying that no actual solution was provided for this issue (already reported several months ago) and that a new stable release of Eclipse and Papyrus will come in 2015 June.

6.2 Extension of network interfaces

The option which has been assessed first for extending <<NetworkInterface>> has been to make that it inherits <<CommunicationMedia>>, which already has the capacity attribute. It is illustrated in Figure 52.

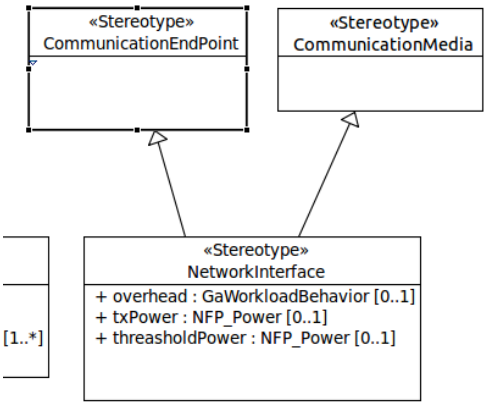


Figure 52 . Tried extension of NetworkInterface for enabling capacity attribute.

However, this extension involves a failure at modelling time in Papyrus 0.8.2. (version associated to Indigo) when entering the stereotype in order to apply values. Then error appearing is reproduced in Figure 53.

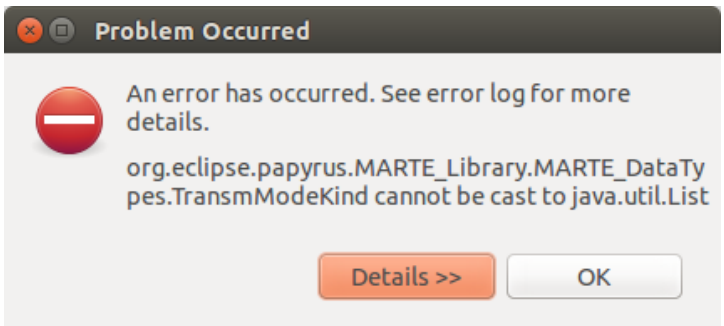


Figure 53. Failure when the extension shown in Figure 52 is tried.

So far, it is to be assessed if this error is a tool related problem or if it correspond to a deeper conceptual problem.

A workaround for this issue consists in enforcing the user to stereotype the network interface component also with <<CommunicationMedia>>. However, this option has been discarded for being less compact and so leading to a heavier modelling procedure. Therefore, the extension and modelling approach shown in section 4.5.2 has been adopted.

7 References

[1] D.Quaglia et al. “CONTREX System modelling methodology (preliminary)”. ”. Deliverable D2.1.1 of the FP7 CONTREX project. Sept. 2014.

[2] J. Medina. “CONTREX System Metamodel”. Deliverable D2.1.1 of the FP7 CONTREX project.

[3] F. Herrera, H. Posadas, P. Peñil, E. Villar, F. Ferrero (GMV), R. Valencia (GMV), G. Palermo "The COMPLEX methodology for UML/MARTE modeling and design-space exploration of embedded systems". Journal of Systems Architecture, V.60, N.1, Elsevier, pp.55–78. 2014-01.

[4] H. Posadas, P. Peñil, A. Nicolás, E. Villar. "Automatic synthesis of communication and concurrency for exploring component-based system implementations considering UML channel semantics" JSA. Accepted.

[5] F. Herrera, H.S. Attarzadeh and I. Sander. "Towards a Modelling and Design Framework for Mixed-Criticality SoCs and Systems-of-Systems". In 16th Euromicro Conference on Digital System Design (DSD). Santander, Sept., 2013.

[6] E. Ebeid, F. Fummi, D. Quaglia, and F. Stefanni, “Refinement of UML/MARTE Models for the Design of Networked Embedded Systems,” in Design, Automation Test in Europe Conference Exhibition (DATE), 2012, Mar. 2012, pp. 1072–1077.

[7] E. Ebeid, D. Quaglia, and F. Fummi. “UML-based Modeling and Simulation of Environmental Effects in Networked Embedded Systems”. In 16th Euromicro Conference on Digital System Design (DSD), 2013.

[8] E. Ebeid, F. Fummi, D. Quaglia. “A Framework for Design Space Exploration and Performance Analysis of Networked Embedded Systems”. In 6th RAPIDO workshop. 2014.

[9] E. Ebedi, F. Fummi, D. Quaglia, J. Medina, P. Peñil, H. Posadas, E.Villar. “Automatic UML/MARTE-based Generation of Multi-domain Simulation Models for Distributed Embedded Systems”. JOURNAL PAPER TO BE SUBMITTED

[10] P. Peñil, A. Díaz, H. Posadas, J. Medina, P. Sánchez. “High-Level design of Wireless Sensor Networks for Performance Optimisation under Security Hazards”. JOURNAL PAPER TO BE SUBMITTED

[11] Á. Díaz, P. Peñil, P. Sánchez, J. Sancho, J. Rico. "Modeling and Simulation of Secure Wireless Sensor networks" Proceedings of the 2012 Forum on Specification and Design Languages, FDL'2012, IEEE. 2012-09

[12] Ebeid E., Medina J., fummi F., and Quaglia D.: “Extensions to the UML Profile for MARTE for Distributed Embedded Systems”. Internal Report Universidad de Cantabria. http://www.ctr.unican.es/reports/I20140522.pdf .

8 APPENDIX A: Summary of Meta-Modelling elements in the background modelling methodologies

8.1 Metamodel from Edalab&U.Verona background

The UML/MARTE methodology developed by Edalab and U.Verona enables the

representation of the:

- Network structure. Such network structure consists on the allocation of network *nodes* and their interconnection via *abstract channels*. It also includes the clustering of the network elements into *zones*, and the capture of *contiguities* among zones.

- Representation of *task* activities.

- Capture of the allocation of task activities into nodes.

- The attributes of the aforementioned elements. It requires the support of the following concepts:

Level	**Modelling Element**	**Meaning**
Application	Task	Basic functionality of the whole application, which reads some input data and produces some output data
	Dataflow	Communication between two tasks
Network/HW resources	Node	Container of tasks with computational resources
	AbstractChannel	Generalization of network channels

Network (environment information)	Zone	Node containers. A zone stablish a cluster of nodes which are always capable to communicate among themselves.
	Contiguity	Relation between two zones. It is used to reflect the environment conditions eventually effecting on the propagation properties among zones.

In turn, each modelling elements requires its own set of attributes:

Modelling Element	**Attributes**	**Type**	**Meaning**
Task	c	ComputationAttr	Computation Resources Required
	m	Boolean	Mobility of the task
	type	TaskType	Type of task modelling
Dataflow	ts	Task	Source task
	td	Task	Destination task
	requiredQoS	QoS	Required QoS
Node	t	Set(Task)	Set of tasks associated to the node
	c	ComputationAttr	Computation Resources provided
	k	NFP_Price	Economic cost of the node
	m	Boolean	Mobility of the node
	p	Power?	Power budget of the node
	gamma	Integer[1]?	Vector of coefficients associated to the tasks assigned, to calculate their contribution to resource (cpu utilization and memory) and power requirements.
AbstractChannel	n	Set(Node)	Connected network nodes
	defaultQoS	QoS	Communication attributes and Tx

			power
	d	LenghtUnitKind	Length of a wire/ maximum range of a wireless link
	k	NFP_Price	Economic cost
	w	Boolean	True means is a wireless channel (if at least a node bound is mobile, it is assumed that w should be true.
Zone	n	Set(Node)	Set of nodes in the zone
	s	Length??	Spatial feature of the zone (surface)
	e		Environment conditions [8]
Contiguity	z1	Zone	First zone
	z2	Zone	Second zone
	Resistance	R	Resistance between zones.

In turn, those attributes require the following data type definitions not present either in UML (as primitive types) or in MARTE.

Enumerated data types:

Data Type	**Value**	**Meaning**
Task Type		Was in which the task is modelled
	CBR	Continuous bit rate
	ON_OFF_CBR	Tasks switchs between sending packets on a CBR mode and non-sending at all (duty cycle, period?)
	Sink	Task consumes instantaneously the packet
	VBR	Variable bit rate (how modelled?)
	User Task	Traffic generation modelled through an activity diagram

Aggregated data types:

Data Type	Attribute	Type	Meaning
ComputationAttr			Computation attributes which reflect either required by a task or provided by a node
	mem_size	Integer	Requirement of memory size (computation resource) of the task
	cpu	Integer	Requirement on utilization of the cpu (computation resource) of the task
CommunicationAttr			Communication attributes which reflect either required by a task or provided by a node
	Throughput	NFP_Frequency	throughput
	Delay	NFP_Duration	End-to-end delay
	Error rate	NFP_Frequency	(packet/bit) error rate
R			Resistance between two contiguous zones
	distance	Real?	
	Throughput_reduction_factor	NFP_Frequency	
	Added_delay	NFP_Duration	
	Added_error_rate	NFP_Frequency	
	Power_reduction_factor	Power	
QoS	c	CommunicationAttr	Quality of Service. It can be required or provided depending on the attribute context. Resulting Communication Attributes
	Pt	Power	Transmission Power

Note that the aggregated attributes present in this table do not need to appear in a profile implementation as separated data types. For instance, while in the profile shown in [1]

appeared as separated elements, in the profile shown in [7], computation and communication attributes appear directly deployed in the task and node modelling elements, and in the dataflow and abstract channel modelling elements respectively.

Note also that, as can be expected, QoS in a network rely much on the values of the communication attributes which can be obtained as the result of the analysis.

Similarly, R type could rely on the QoS type, all attributes except for distance refer to a representation or QoS reduction.

8.2 Meta-model from UC background

A component-based methodology which supports separation of concerns supporting views for the detailed description of the node internals (application, SW architecture, HW architecture, etc) and views for the description of the network with nodes and the network.

Morever, the following modelling elements are required:

Level	Modelling Element	Meaning
Application / SW / HW	Data types, interfaces, RtUnits, HW components, …	A detailed description of the internal architecture and attributes of the MPSoC. It includes the architecture of the application, software and hardware components, and specification of allocations at multiple levels
HW Architecture	HWbattery	
	Hwsensor	Reflects the IO controller of the HW architecture in charge of sensing a specific physical parameter of the environment
	Network interface	Reflects the network interface element in charge of connecting the MPSoC node resources with the network infrastructure
Network	Node	Node
	Node-to-Node link	Description of the wireless connection of the

In addition, the following attributes are required for the modelling elements:

Modelling Element	Attribute	Type	Meaning
Network Interface and Node-to-Node link			Computation attributes which reflect either required by a task or provided by a node
	trans_power	NFP_Power	Transmission power used to transmit a packet
	retries	Integer	Number of retrials required for transmission
	encripted	Boolean	Specifies if packets are encrypted for transmission
Node-to-Node	prob	Real	Communication success

link			probability

9 APPENDIX B: Features under discussion

This appendix includes additional features that are or have been under discussion, but not introduced in the methodology.

9.1 Modelling of Routing Paths

The functionality of application components states the precise destination address of the packages. However, abstract node models shown in sections 4.5.6 and 4.5.7 state nothing to that regard. Default semantics of random addressing of the packets and routing in the nodes is assumed.

If a modelling scenario requires the modelling of a specific addressing of packets, and of fixed routing paths, in a model with abstract nodes, then the model shall include additional information capturing such addressing and fixed rooting.

A possible solution is to use a communication diagram to capture such an information. It is illustrated in Figure 54.

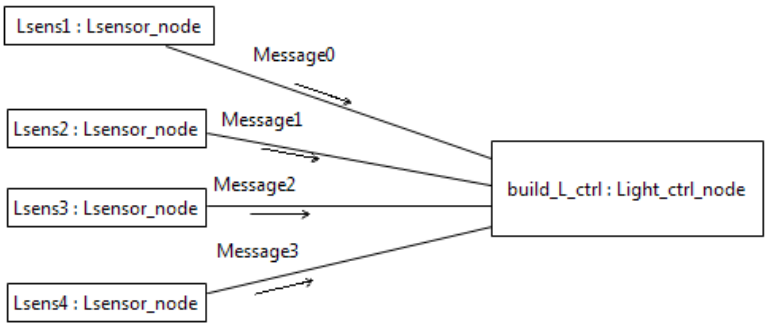


Figure 54. A communication diagram can be used to specify the traffic flow between source and sink nodes.

Figure 54 diagram enables the definition of destination addresses. However, some performance models can still require a more precise modelling of the path traversed by packets within the network. The methodology enables a further degree of precision by stating complete and fixed routing paths between source and sink nodes. It requires referring all the instances reflected in the network architecture which are involved in the specified routing paths. Figure 55 provides an example based on Figure 30 network example.

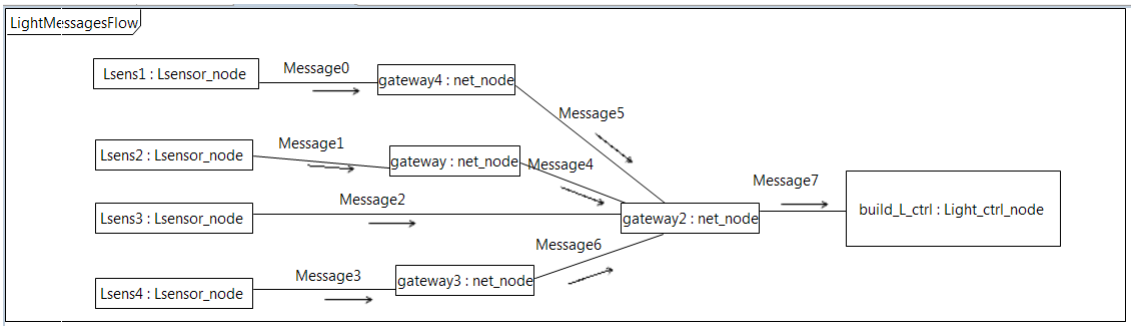


Figure 55. Precise definition of the routing paths followed by the packet traffic modelled though abstract models of nodes and sources.