ESD Electronic Systems Design

Synthesis Verification Testing Power Communication

Toolchain for Network Synthesis

Alex Malfatti, Davide Quaglia

A.A. 2014/2015

Outline

- Introduction
  - Network Synthesis
  - CASSE
- Methodology
  - High-level description
  - Intermediate description
  - Simulation model
- Toolchain
  - Tools
- Exercises

Introduction

Network Synthesis

- Network synthesis is a design process which starts from a high-level specification of a distributed system and finds an actual description of its communication infrastructure in terms of mapping of application tasks onto network nodes, their spatial displacement, the type of channels and protocols among them, and the network topology.

CASSE (1)

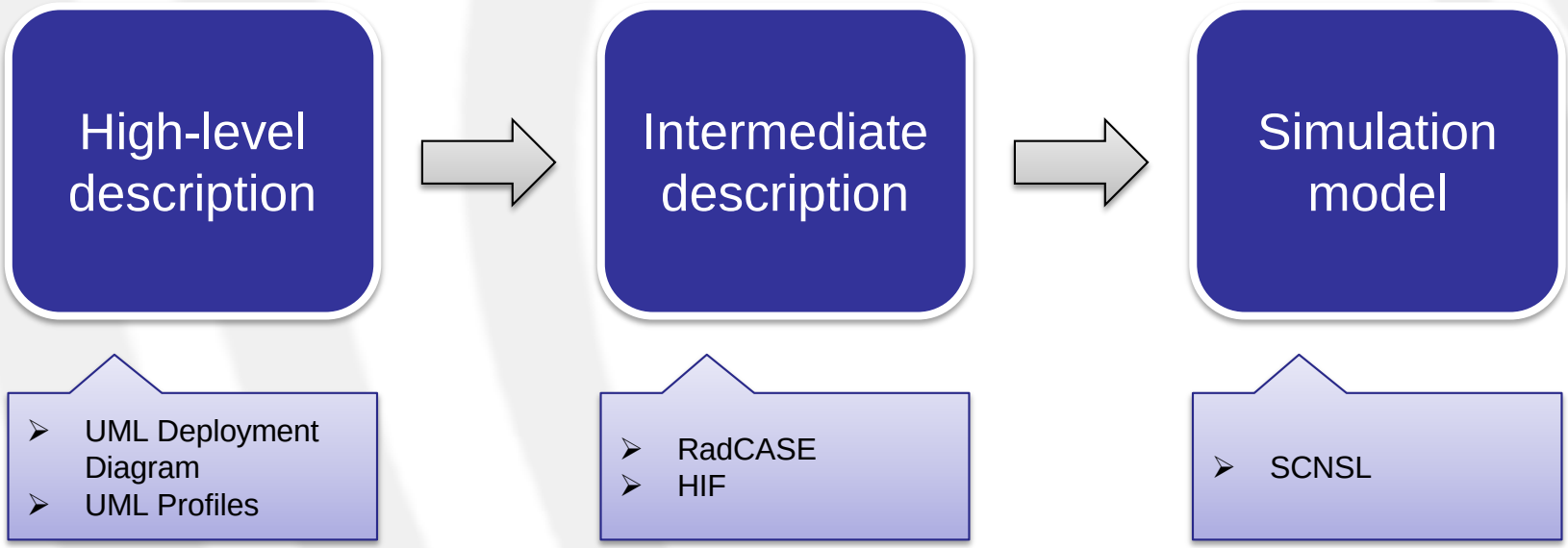
- Communication Aware Specification and Synthesis Environment (CASSE), is an extended design flow, which addresses the network synthesis, in terms of nodes, tasks, data flows, abstract channels, zones, and contiguities.
  - Tasks
    - A task represents a basic functionality of the whole application; it takes some data as input and provides some output.
  - Data flows
    - A data flow (DF) represents communiction between two tasks; output from the source task is delivered as input for the destination task.
  - Nodes
    - A node can be seen as a container of tasks.

CASSE (2)

- Abstract Channels
  - An abstract channel (AC) interconnects two or more nodes; referring to the ISO/OSI model and assuming that the functionality to be designed is at level N, the AC contains the physical channel, and all the protocol entities up to level N-1.
- Zones
  - A zone is a partition of the space which contains nodes; each zone is characterized by an environmental attribute.
- Contiguities
  - Zones are related by the notion of contiguity defined as follows:
    - Two zones are contiguous if nodes belonging to them can communicate each other;
    - Contiguity represents not only the physical distance between zone, but it can be used also to model environmental obstacles, like walls.

Methodology

Methodology



- The methodology starts from a high-level description of the network scenario in which a communication infrastructure is already defined.
- The final result is a simulation model of the considered scenario.

High-level description (1)

- UML Deployment Diagrams
  - The deployment diagrams represents the physical deployment of processes (named *Artifacts*) on containers (named *Nodes*) connected through UML *Communication paths*.
  - Other UML entities can be used in deployment diagrams, i.e., *Devices* to model HW nodes, *Packages* to group nodes, *Dependencies* to connect either Artifacts or Packages together.
- UML Profiles
  - The profiles extend the UML semantics.
  - The are defined by *stereotypes*, *tag definitions*, and *constraints* that are applied to specific model elements.

UML Classifier	NW Entity
Artifact	Task
CommunicationPath	Point-to-point Channel
Dependency	DataFlow/Contiguity
Device	Shared Channel
Node	Node
Package	Zone

High-level description (2)

UML to NW - *Instances*

UML	NW
Node ← Artifact	Node ← Task
CommunicationPath ← Node	Point-to-point Channel ← Node
Device ← Dependency ← Node	Shared Channel ← Node
Package ← any UML Classifier	Zone ← any NW Entity
Dependency ← Artifact	DataFlow ← Task
Dependency ← Zone	Contiguity ← Zone

High-level description (3)

UML to NW - *Bindings*

Intermediate description

- RadCASE
  - radCase is an integrated tool suite for developing, testing and integrating embedded software applications based on Model Driven Architecture approach.
- Heterogeneous Intermediate Format (HIF)
  - HIF is a HW/SW description language structured as XML tree of classes. Each class describes a specific component or functionality that is tipically provided by hardware description languages (e.g., VHDL, Verilog, and SystemC).
  - It provides designers with a convenient way to automatically manipulate HW/SW descriptions.

Simulation model

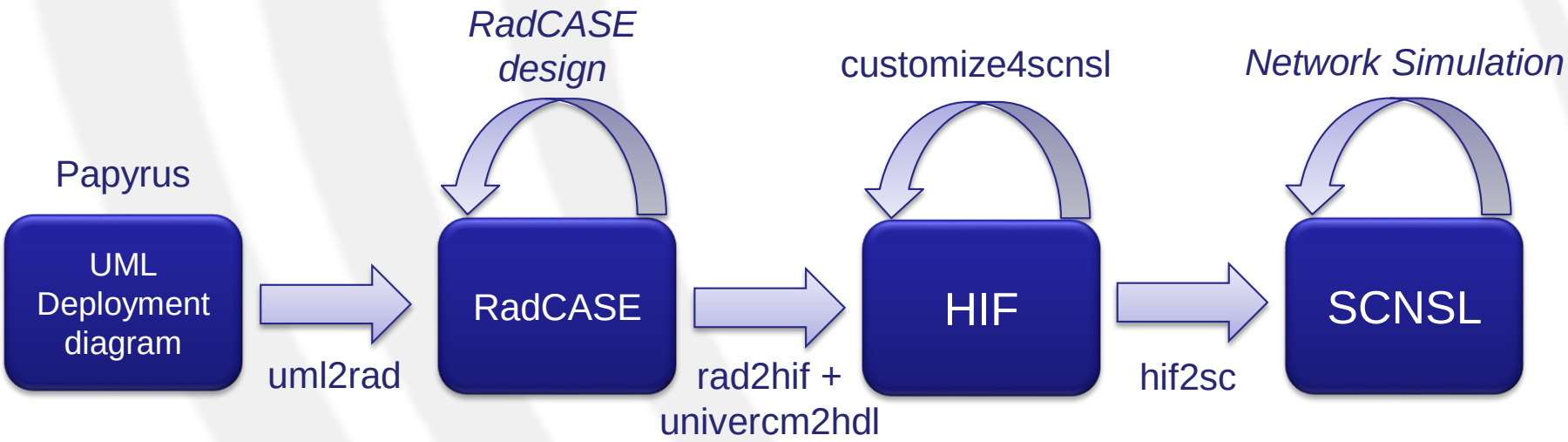
- SystemC Network Simulation Library (SCNSL)
  - SCNSL is an extension of SystemC to allow modelling packet-based networks such as wireless networks, Ethernet, and fieldbus.
  - SCNSL provides primitives to model packet transmission, reception, contention on the channel and wireless path loss.
  - The use of SCNSL together with SystemC allows the easy and complete modeling of distributed applications of networked embedded systems such as wireless sensor networks, routers, and distributed plant controllers.

Not yet implemented in the current methodology

- Broadcast transmission
- Directionality specification for Point-to-point Channels (CommunicationPath in UML Deployment Diagram)
  - Only «halfDuplex» or «fullDuplex» transmission modes are available for point-to-point channels
- Protocols definition
  - Communicators in SCNSL will not be automatically created

Toolchain

UML2SCNSL Toolchain

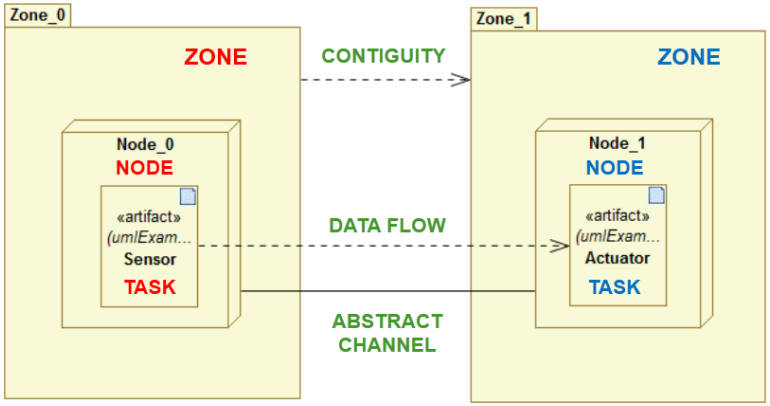


Tools

- Papyrus
- HIFSuite
  - uml2rad
  - rad2hif + univercm2hdl
  - customize4scnsl
  - hif2sc
- SCNSL

Tools (1)

- Papyrus
  - Papyrus is an Eclipse plug-in tool for editing UML and generating a textual description (e.g., *XML*) from UML models.
  - Papyrus provides a user-friendly and free editor for modeling of *UML Deployment diagram*, supporting profiles, such as MARTE and the *Network profile*.



Tools (2)

- uml2rad
  - `uml2rad` is a tool which converts UML descriptions, obtained from Papyrus, in a *radCASE* representation; this description can be manipulated to implement functional parts of the network (e.g., tasks implementation).
- rad2hif + univercm2hdl
  - `rad2hif` and `univercm2hdl` are tools whose combined usage generates an *HIF* representation from the radCASE one; the final description obtained includes all the informations about network instances and bindings between them.

Tools (3)

- customize4scnsl
  - `customize4scnsl` is a tool which analyzes the HIF description obtained in this point of the toolchain, and generates a new HIF representation compliant with SCNSL specifics.
- hif2sc
  - `hif2sc` is a HIFSuite back-end tool which converts an HIF description into SystemC code.
  - In our case the result of the translation is a SCNSL description of the Network infrastructure and, eventually, of its functionality.
- SCNSL

Exercises

Setup Exercises

- In order to setup the tools for the exercises, go to the "*/tmp/*" directory and untar the «*exercises_nesLab3.tar.gz*».

```
1$ tar -xzvf exercises_nesLab3.tar.gz
```

- "*exercises_nesLab3*" contains the following directories:
  - *Tools*
    - *hifsuite* → set of tools for the "*uml2scnsl*" toolchain
    - *Papyrus* → Eclipse plug-in to create/edit UML models
  - *uml2scnsl*
    - 0.UML_Examples → where to put the «*.uml*» files
    - 1.uml2rad → files generated by `uml2rad` tool
    - 2.rad2hif → files generated by `rad2hif` tool
    - 3.univercm2hdl → files generated by `univercm2hdl` tool
    - 4.customize4scnsl → files generated by `customize4scnsl` tool
    - 5.hif2sc → files generated by `hif2sc` tool
    - 6.scnsl → SCNSL source codes

Setup Exercises - HIFSuite

- Move to the "*Tools/*" directory, inside the unpacked directory.

```
1$ cd Tools
```

- Export in the environment the paths to HIFSuite tools and SystemC. A script, namely "*env-setup.sh*", is provided to help this task: edit it (e.g., emacs, gedit, etc.) and then source it.

```
1$ emacs ./hifsuite/env-setup.sh
2$ source ./hifsuite/env-setup.sh
```

  - In the script you have to replace *<HIFSUITE_PATH>* with your current path to the hifsuite directory (e.g., "*/tmp/exercises_nesLab3/Tools/hifsuite*").

Setup Exercises - Papyrus

- Run the Windows 7 Virtual Machine and copy wherever you want the *Papyrus* directory.
- Launch "*Papurus.exe*".
  - Ensure that Java Runtime Environment (JRE) 32 bit is installed.
- Create the workspace inside the folder shared with Linux:
  - *"//10.0.2.4/qemu/workspace"*

Setup Exercises – uml2scnsl

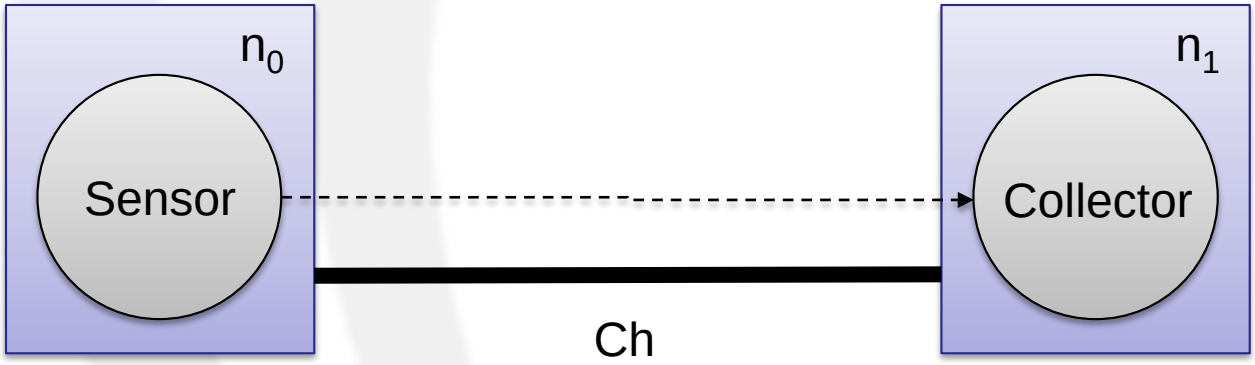
- In the "*uml2scnsl/*" directory there is a script, namely "*uml2scnsl.sh*", which takes as parameter the name of the «*.uml*» file representing the UML model of the Network Scenario.

```
1$ ./uml2scnsl.sh NWScenario_name
```

  - Before to run the script, you have to modify it replacing *<HIFSUITE_PATH>* with your current path to the hifsuite directory, as done before.

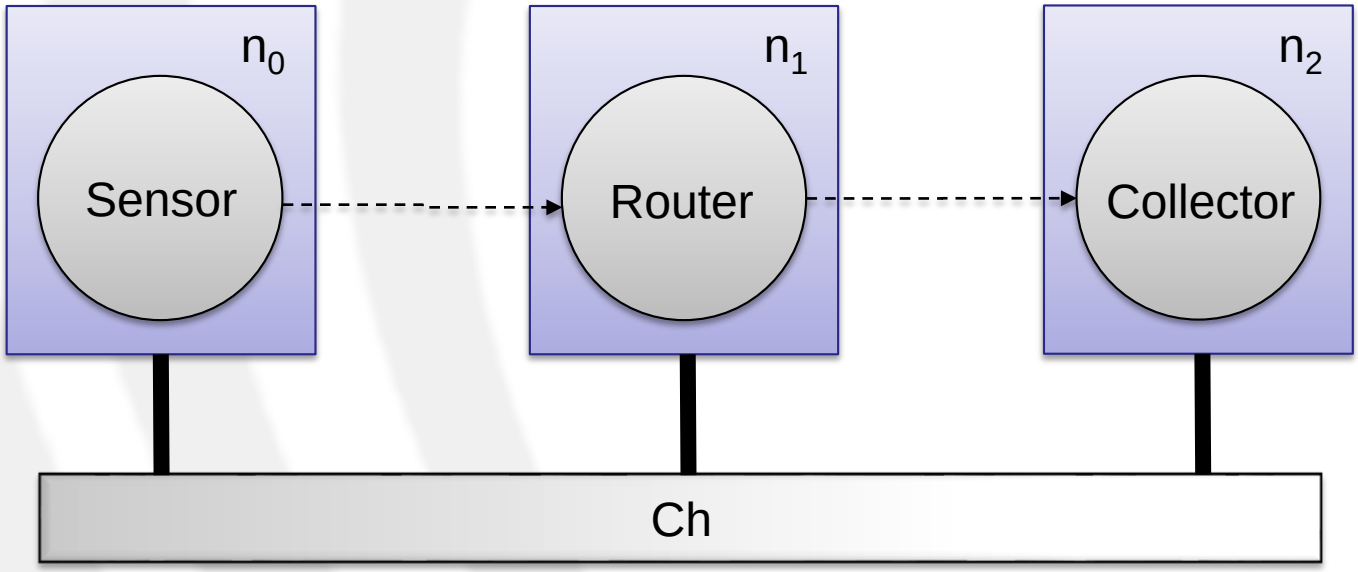
- When the script has finished its execution, the resulting scnsl source code is putted in the corresponding scnsl directory (e.g., "*6.scnsl/NWScenario_name/src/hif_globals.cc*")

Exercise 1: Two Nodes



1. Model the scenario above in UML by using Papyrus + NW Profile.
   - Sensor → «sensing» behavior
   - Collector → «collection» behavior
   - Ch → «*fullDuplex*» point-to-point channel
   - n_0 and n_1 in two distinct zones
2. Generate the corresponding SCNSL code by using the *uml2scnsl* toolchain.
3. Compile and execute it.

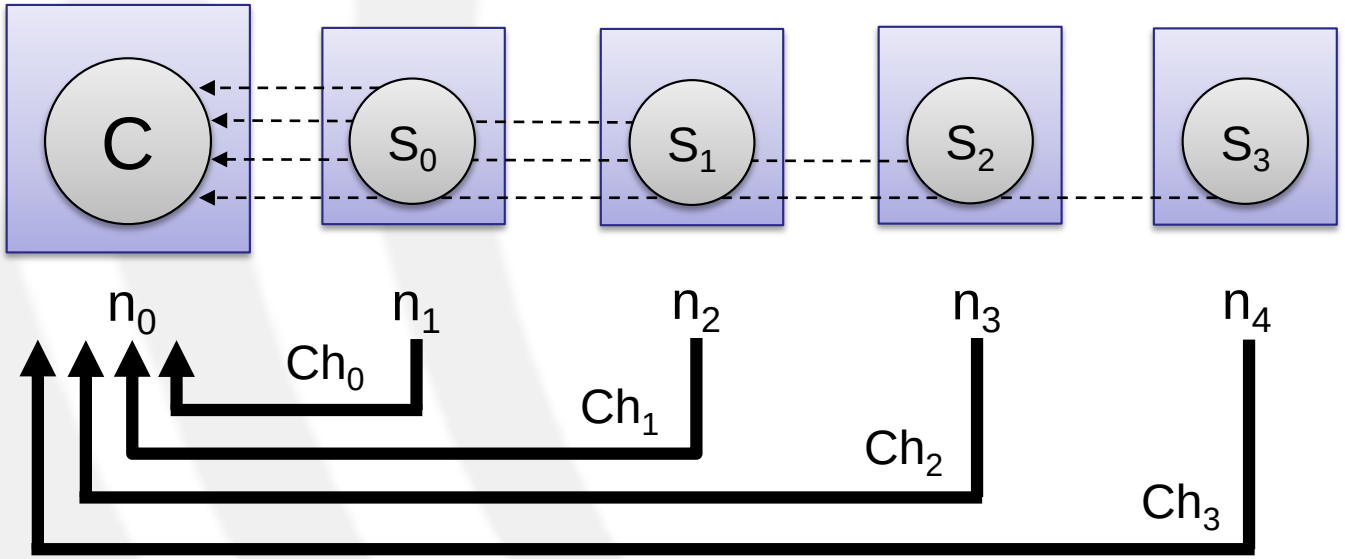
Exercise 2: Three Nodes with Router



1. Model the scenario above in UML by using Papyrus + NW Profile.
   - Sensor → «sensing» behavior
   - Router → «routing» behavior
   - Collector → «collection» behavior
   - Ch → shared channel
   - n_0 and n_1 in a distinct zone with respect to n_2
2. Generate the corresponding SCNSL code by using the *uml2scnsl* toolchain.
3. Compile and execute it.

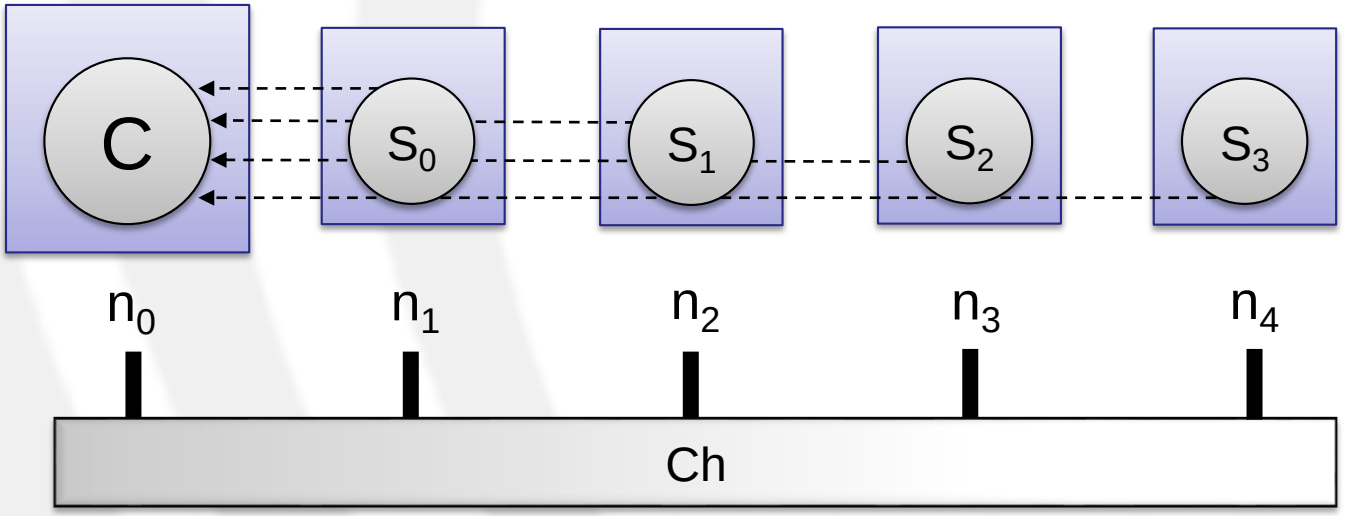
Hint: in Papyrus use Dependencies to link Nodes with Devices (wireless channels).

Exercise 3: Temperature monitoring for Building Automation (1)



1. Model the scenario above in UML by using Papyrus + NW Profile.
   - c → «control» behavior
   - s_n ($0 \le n \le 3$) → «sensing» behavior
   - Ch_n ($0 \le n \le 3$) → «*fullDuplex*» point-to-point channel
   - Each node in a distinct zone
2. Generate the corresponding SCNSL code by using the *uml2scnsl* toolchain.
3. Compile and execute it.

Exercise 3: Temperature monitoring for Building Automation (2)



4. Change the current point-to-point transmission to a shared one in the UML description.
   - Ch → shared channel
5. Generate the corresponding SCNSL code by using the *uml2scnsl* toolchain.
6. Compile and execute it
   - What can you say about the Packet Loss Rate (PLR)? Why?