ESD Electronic Systems Design
Synthesis Verification Testing Power Communication

# Toolchain for Network Synthesis

Enrico Fraccaroli, Davide Quaglia
Department of Computer Science
University of Verona


A.A. 2015/2016

# Outline

- Introduction
  - Network Synthesis
  - CASSE
- Methodology
  - High-level description
  - Resulting Synthesized Network
- MILP
  - Variables
  - Constraints
  - Objectives
- Workspace Setup
  - Install Gurobi
  - Get the lab source code
  - Execute the Network Sythesizer
  - Write High-Level Description
- Exercises

# Introduction

# Network Synthesis

- Network synthesis is a design process which starts from a high-level specification of a distributed embedded system and finds an actual description of its communication infrastructure in terms of mapping of 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 a formal description, which supports the network synthesis.
- It defines:
  - Tasks
  - Data Flows
  - Nodes
  - Abstract Channels
  - Zones
  - Contiguities

# CASSE (2)

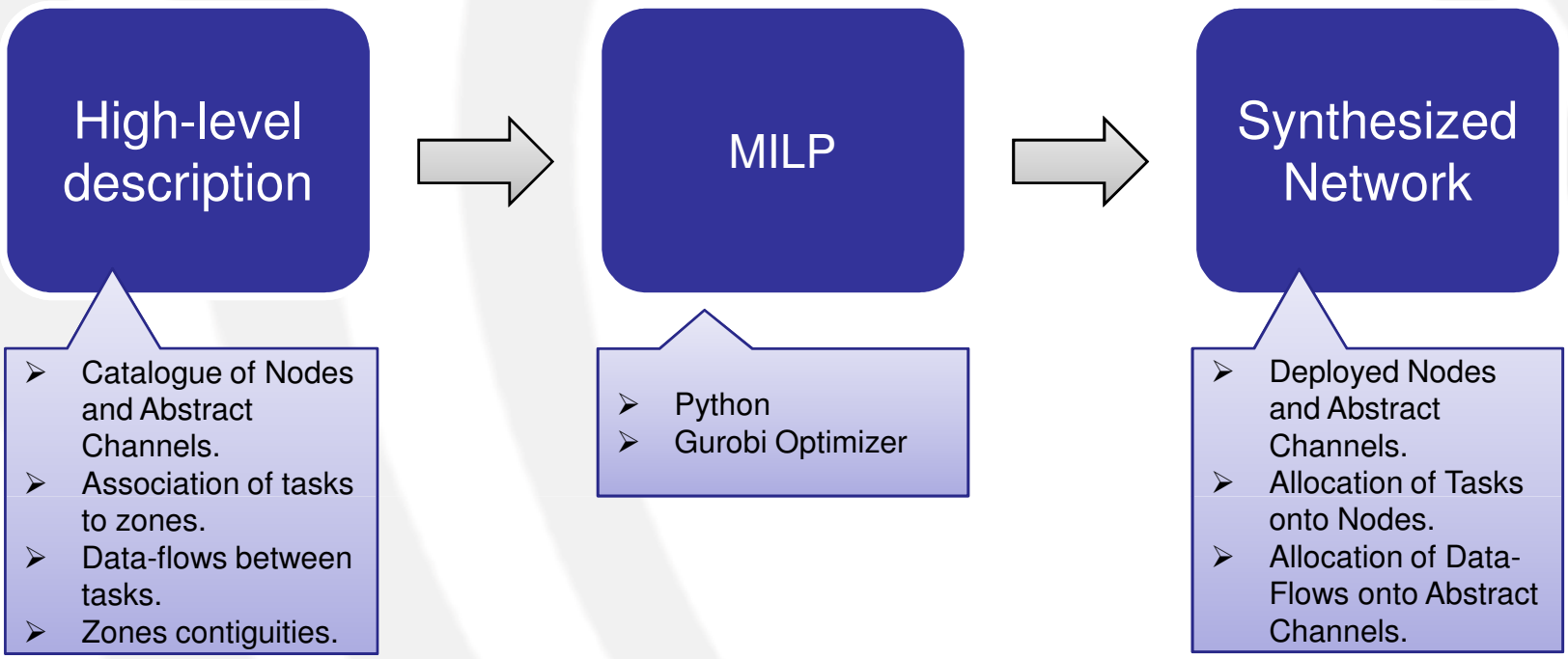
- Tasks
  - A task represents a **basic functionality** of the whole application; it takes some data as input and provides some output.
- Data flows
  - A data flow (DF) represents **communiction** between two tasks; output from the source task is delivered as input to the destination task.
- Nodes
  - A node can be seen as a **container** of tasks.

# CASSE (3)

- 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 according to tasks distribution**.
- 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.
    - Such influence over the communication is modelled in terms of a coefficient which depends on the **zone pair** and the **type of abstract channel**.

# Methodology

# Methodology




- The methodology starts from a **high-level description** of the distributed embedded system, which is **implementation-independent**.
- The final result is a **synthesized network infrastructure** which can be used for the generation of both a simulation model and the actual deployment.

# High-level Description

- Node and Channel Catalogs
  - **Node Catalog**
    - Contains the possible types of nodes.
  - **Abstract Channel Catalog**
    - Contains the possible types of channels.
- Input Instance
  - Reports all the details concerning the problem.
  - It contains:
    - Set of **Tasks** inside **Zones**.
    - Set of **Data-Flows**.
    - Set of **Contiguities** between **Zones**.

# Resulting Synthesized Network

- A feasible solution consists in:
  - How many **nodes** of each type should be placed in each zone.
  - Which of the instantiated nodes will host the **tasks**.
  - How many **abstract channels** of each type should be deployed.
  - Which of the activated abstract channels will host the **data-flows**.

*Mixed-Integer Linear Program*

# MILP

# MILP

**Integer Linear Program (ILP)**

- An optimization model is an **Integer Linear Program**.
- If all of its variables are discrete, the model is a **pure integer linear program**.
- Otherwise, the mode is a **mixed-integer linear program**.

- The problem which are most commonly solved are of the form:
  - **Objective**
    - **minimize** $c^T x$ (**linear** cost function)
  - **Constraints**
    - $A\,x \leq b$ (**linear** constraints).
    - $x \geq 0$ (**bound** constraints).
    - some or all $\mathbf{x}_j$ must take integer values (**integrality** constraints).

# Variables

- Integer Variables:

$N_{n,z}$ How many nodes of type **n** are deployed inside zone **z**.
The *upperbound* on the number of active nodes is: $\bar{N}_{n,z}$

$C_c$ Hown many channels of type **c** are activated.
The *upperbound* on the number of active channels is: $\bar{C}_c$

- Binary Variables:

$x_{n,z,p}$ At least **p** nodes of type **n** allocated in zone **z**.

$y_{n,z,p}$ At least **p** channels of type **c** deployed.

$\gamma_{d,t}$ Data-flow **d** tasks and taks **t** are placed in three different nodes.

$\rho_{t1,t2}$ Tasks **t1** and **t2** are mapped into different nodes.

$m_d$ One of the tasks of data-flow **d** has a mobility requirements.

$w_{t,n,p}$ Task **t** is placed inside the **p-th** node of type **n** in its zone.

$h_{d,c,p}$ Data-flow **d** is placed inside the **p-th** channel of type **c**.

# Constraints

- *Eg.* Number of Instantiated Components
- The number of active nodes ($N_{n,z}$) is equal to the sum of all the actually instantiated nodes ($x_{n,z,p}$):

$$N_{n,z} = \sum_{p=1}^{\bar{N}_{n,z}} x_{n,z,p}$$

- The same applyes for the channels:

$$C_c = \sum_{p=1}^{\bar{C}_c} y_{c,p}$$

- *Eg.* Nodes Capacity
- Given a node *n* the sizes of the tasks placed inside the node has to comply withe the available space of the node:

$$\sum_{t=a_t(n)} t_s * w_{t,n,p} \leq n_s$$

# Objectives

- Our goal is to minimize or maximize an objective function value.
- For what concerns the network synthesis problem, one could choose to Minimize the Economic Cost of the synthesized network.
- Such objective can be specified as follows:

$$min\left[\sum_{n\in N}\sum_{z\in Z}\sum_{p\in \bar{N}_{n,z}}\left(x_{n,z,p} * n_{cost}\right) + \sum_{c\in C}\sum_{p\in \bar{C}_c}\left(y_{c,p} * c_{cost}\right)\right]$$

  - We've basically summed the costs of all the actually instantiated nodes to the sum of all the actually instantiated channels.

# Workspace Setup

# Install Gurobi (1)

- Now we will see the needed steps to get a free version of Gurobi-Optimizer and how to set up the experiments:

1. First, you have to register at:
   - http://www.gurobi.com/registration/general-reg
   - Select as Account Type: **Academic.**
   - At the end of the registration process you will receive a mail.
2. Open the received mail:
   - Inside the mail you will found a link which will allow you to set a password for your Gurobi account.
3. Download gurobi-optimizer at:
   - http://user.gurobi.com/download/gurobi-optimizer.
   - Select **Linux 64** and then press **Download**.
4. Move the downloaded compressed file inside your home directory.

# Install Gurobi (2)

5. Untar the compressed file (*replace X.Y.Z with the identifier of your downloaded version*):

```
tar xvzf gurobiX.Y.Z_linux64.tar.gz
```

6. Rename the uncompressed directory:

```
mv gurobiXYZ ~/Gurobi
```

7. Create a script:

```
gedit ~/set-gurobi-env.sh
```

8. Place the following commands inside the script:

```
export GUROBI_HOME="${HOME}/Gurobi/linux64"
export PATH="${PATH}:${GUROBI_HOME}/bin"
export LD_LIBRARY_PATH="${LD_LIBRARY_PATH}:${GUROBI_HOME}/lib"
export GRB_LICENSE_FILE="${GUROBI_HOME}/gurobi.lic"
```

9. Make the script executable:

```
chmod +x ~/set-gurobi-env.sh
```

10. Execute the script:

```
source ~/set-gurobi-env.sh
```

# Install Gurobi (3)

11. Get a free academic license at:
    - http://www.gurobi.com/download/licenses/free-academic
12. Copy the command at the end of the of the page, the one which has the following form

```
grbgetkey xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
```

13. Paste the command inside your Bash.
14. When prompted set the destination folder to your GUROBI_HOME. So if you've followed the instruction just type:

```
~/Gurobi/linux64
```

# Install Gurobi (4)

15. Move inside GUROBI_HOME:

```
cd $GUROBI_HOME
```

16. Install Gurobi inside a directory of your own:

```
python setup.py install --prefix=~/GurobiLib
```

17. Lets define a new environment variable which points to Gurobi library (replace X.Y with your Python version which can be read when you launch Python interpreter):

```
export GUROBI_LIB=${HOME}/GurobiLib/lib/pythonX.Y/site-packages/gurobipy
```

18. Let python know where the library is:

```
export PYTHONPATH=${PYTHONPATH}:${GUROBI_LIB}
```

19. You can place the previous two exports inside your set-gurobi-env.sh script and re-source it.

# Get the Lab Source Code

- The source code for the today's laboratory is available at:

```
https://bitbucket.org/Shalander/public_nes_laboratory_univr
```

- In order to retrive the source code you have to clone the repository.
- **Create** and **move** into a directory inside your home:

```
mkdir ~/LabNes;
cd ~/LabNes;
```

- **Clone** from the repository the source code:

```
git clone https://Shalander@bitbucket.org/Shalander/public_nes_laboratory_univr.git .
```

**GIT**

A version controll system mainly used for software development.

# Execute Network Synthesizer

- Move inside the directory which contains the source code:

```
cd ~/LabNes/LabGurobi
```

- Execute the synthesizer using the provided script which requires the following arguments:

```
./Synthesize.sh <Test_Case_Directory> <Optimization Objective>
```

- So, you can try with:

```
./Synthesize.sh TestCase1 1
```

- This will synthesize the test case contained inside the provided folder and will optimize it w.r.t. the **economic cost minimization** objective.

# Write High-Level Description (1)

- A Test Case contains three files:
  1. input.txt
  2. nodes.txt
  3. channels.txt
- Each input file underlies the following notation:
  - A row which starts with a '**#**' or '**;**' is a comment and therefore ignored.
  - The first (non-comment) row is the header of the file which contains the counter of the contained items.

# Write High-Level Description (2)

- File: '**input.txt**'

```
# Zones | Contiguites | Tasks | Data-Flows
  2       2             2       1
# Contiguities
# Zone1 | Zone2 | Channel | Conductance
  1       2       1         0.7
  1       2       2         0.9
# Tasks
# Label | Size | Zone | Mobile
  Tsk1    3      1      0
  Tsk2    7      2      1
# Data-Flows
# Label | Source | Target | Band | Delay | Error
  Df01    Tsk1     Tsk2     4      10      4
```

- File: '**channels.txt**'

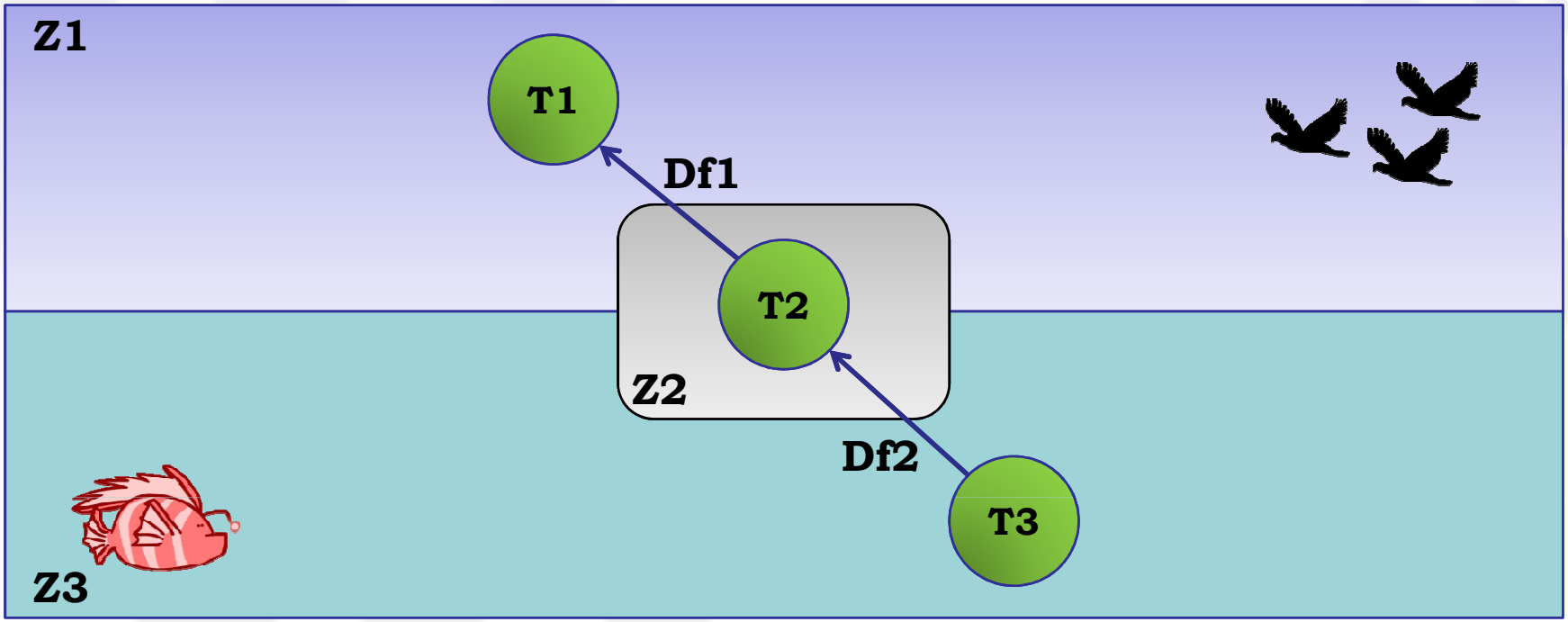
```
# Number of channels:
  2
# Label     | Id | Cost | Size | Wless | Power | Delay | Error
  Bluetooth   1    5      10     1       15      5       5
  Fiber       2    25     30     0       5       1       1
```

- File: '**nodes.txt**'

```
# Number of nodes:
  2
# Label      | Id | Cost | Size | Mobile | Power | TaskPower
  board        1    5      10     0        5       4
  smartphone   2    25     30     1        25      8
```

# Exercises

# Exercise One

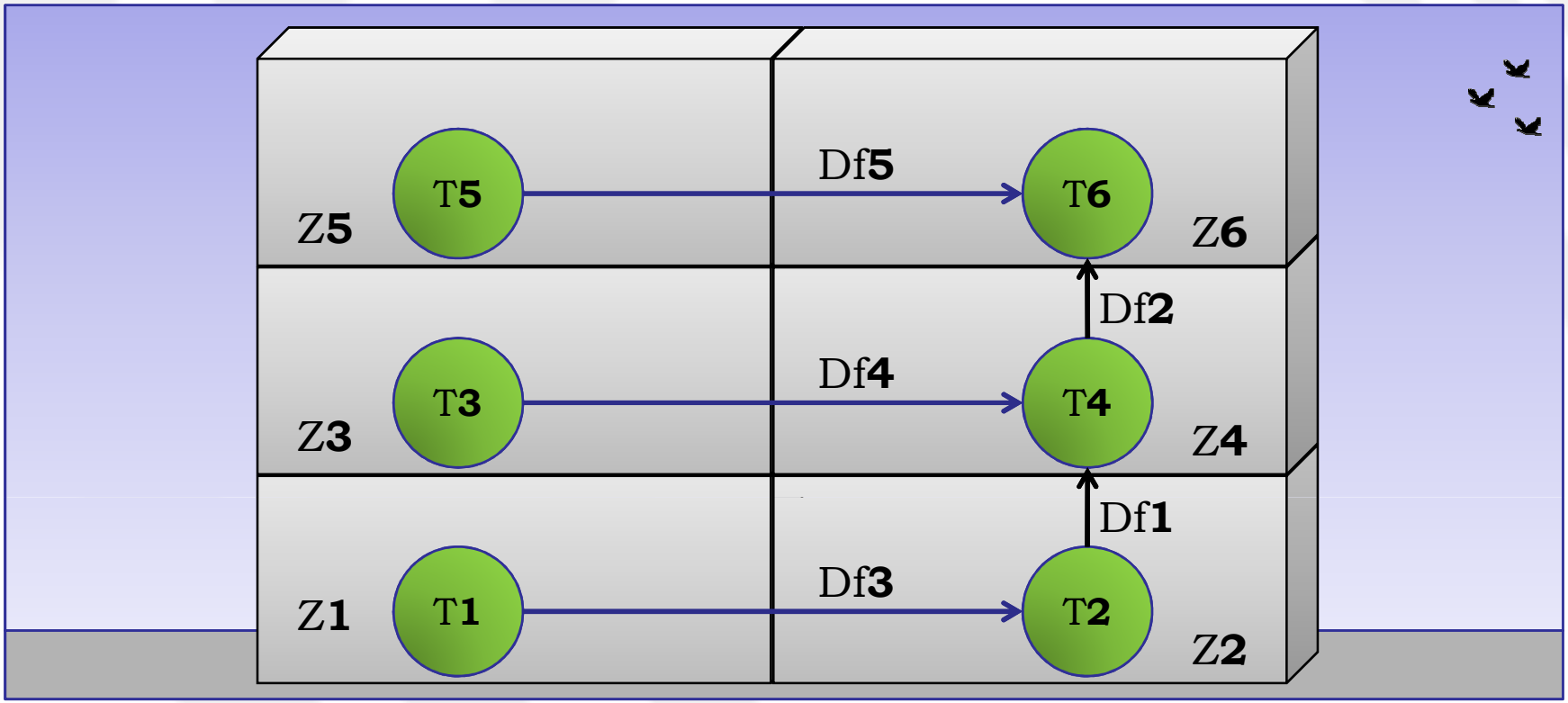



- Model the scenario above using the presented formalism:
  - Tasks:
    - [**T1**:**s**=10; **m**=1;], [**T2**:**s**=25; **m**=1;] [**T1**:**s**=5; **m**=1;]
  - Data-Flows:
    - [**Df1**:**src**=T2; **dst**=T1; **b**=7; **maxD**=5; **maxE**=15],
    - [**Df2**:**src**=T3; **dst**=T2; **b**=25; **maxD**=15; **maxE**=3]
- Choose properly the **contiguities**, **nodes**, **channels**.

# Exercise Two

1. Open the network synthesis script (*i.e.*, Synthesizer.py).
2. Localize variable **h** declaration. Such variable determines the allocation of data-flows inside specific instances of a given type of channel.
3. Change the type of the variable (*i.e.*, attribute vtype) from GRB.**BINARY** to GRB.**CONTINUOUS**.
4. Try to synthesize a network (*e.g.*, test case 2 with objective 1).
5. Try to **understand** the reason for what happens when you change the type of such variable.

# Exercise Three




- Model the scenario above using the presented formalism:
  - [**T2**,**T4**,**T6**:**s**=25; **m**=0]; [**T1**,**T2**,**T5**:**s**=5; **m**=1]
  - [**Df1**, **Df2**:**b**=30; **d**=10; **e**=10]; [**Df3**, **Df4**, **Df5**: **b**=10; **d**=5; **e**=5]
- Choose properly the **nodes** and **channels**.
- Set the **contiguities** considering that radio waves do not propagate between floors but only between rooms on the same floor
  - only **wired** channels can be used between Z**2** and Z**4** as well as between Z**4** and Z**6**.