



# Design flow for Networked Embedded Systems

**Emad Ebeid**  
Ph.D. student @ CS depart  
University of Verona, Italy  
Emad.Ebeid@univr.it

**Davide Quaglia**  
Assistant Professor @ CS depart  
University of Verona, Italy  
Davide.Quaglia@univr.it



## Outline

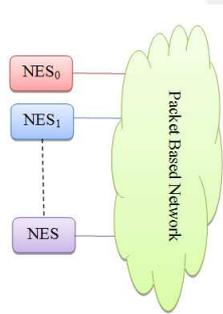
- Introduction and motivation
- Background
- Proposed methodology
- Modeling requirements
- System view simulation
- Network synthesis
- Network view simulation
- Case study
- Conclusion

2

**ESD** Embedded Systems Design

## Introduction

- **Networked Embedded Systems (NES)** are an important class of devices
  - Network functionalities are at the core of design objectives
  - Network requirements come together with traditional requirements
- **Distributed Embedded Systems** are group of NES which are connected together using network interfaces, standardized protocols and channels
  - Example: Temperature control of a building



The diagram shows three rectangular boxes representing Networked Embedded Systems (NES). The top box is labeled NES<sub>0</sub> (red), the middle is NES<sub>1</sub> (blue), and the bottom is NES (purple). They are connected to a central green cloud labeled 'Packet Based Network'. Dashed lines indicate the network connections between the nodes and the network cloud.

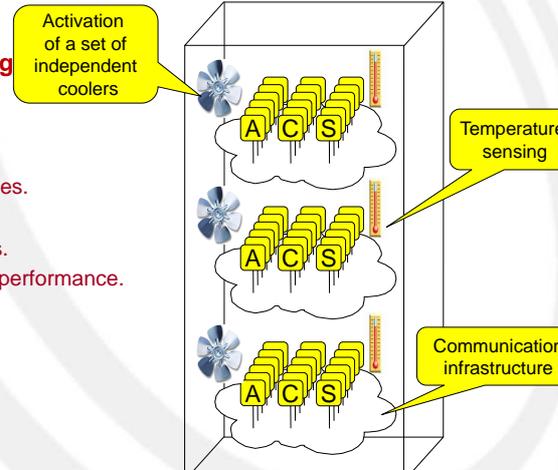
3

**ESD** Embedded Systems Design

## Introduction

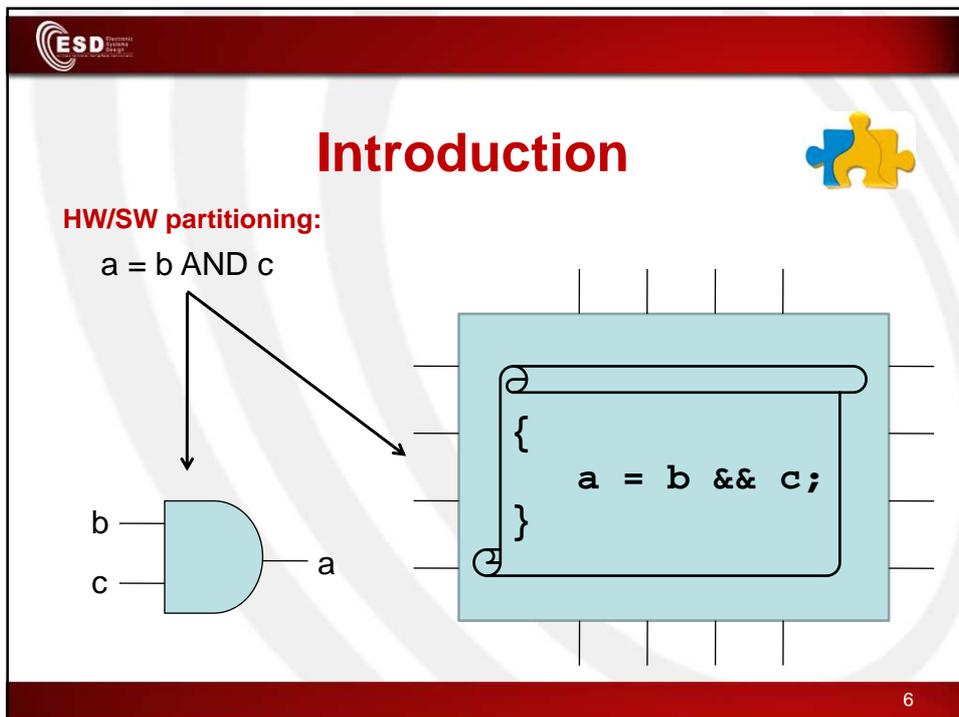
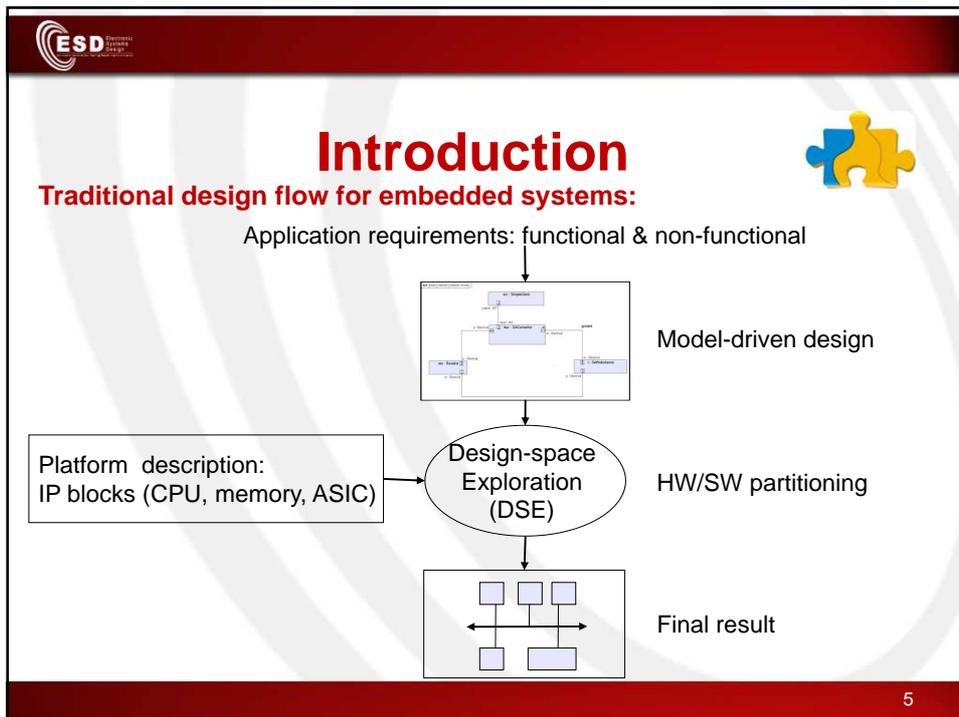
### Temperature control of a building

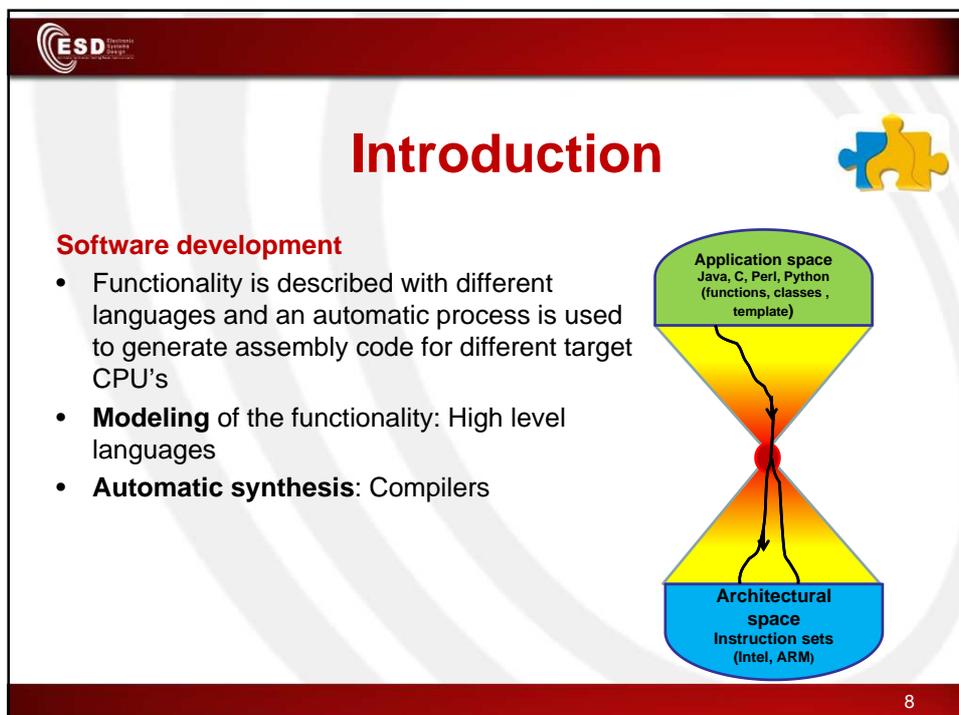
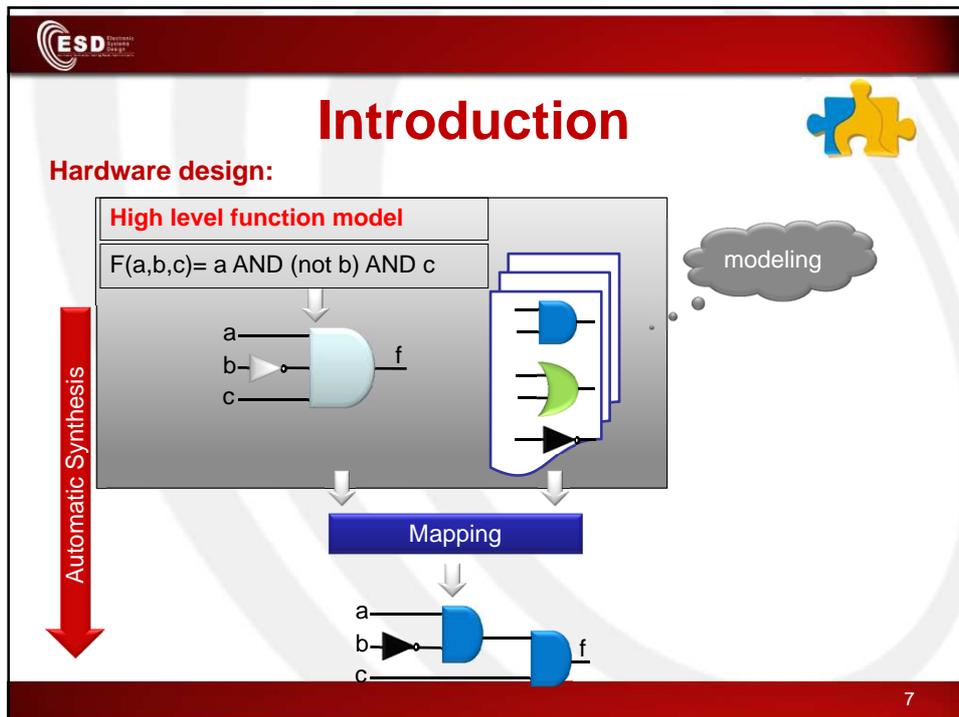
- Scenario:
  - Hundreds of concurrent tasks.
  - Heterogeneous tasks.
  - Devices with different capabilities.
  - Wireless and wired channels.
  - Many communication protocols.
  - Nodes position affects system performance.
- Questions:
  - How many nodes?
  - How to assign tasks to nodes?
  - Which network protocols?
  - Which intermediate systems?



The diagram illustrates a building with three floors. Each floor has a set of nodes labeled 'A', 'C', and 'S'. Callouts point to specific features: 'Activation of a set of independent coolers' points to a fan icon on the top floor; 'Temperature sensing' points to a thermometer icon on the middle floor; and 'Communication infrastructure' points to a cloud-like network structure at the bottom of the building.

4





**ESD** Elements System Design

## Introduction

- **Distributed embedded application as a single system to be designed**

9

**ESD** Elements System Design

## Introduction

### New design flow for NES

□ Steps of state-of-the-art system design flow

■ Additional steps for network design

Application requirements

10



## Introduction

- Start from an abstract Model-Based System Specification
- Modeling and Analysis of Real-Time and Embedded Systems (MARTE) profile for the unified modeling language (UML)
- Refinement steps and simulations
- Standard representation of requirement and solutions



11



## Background

- Design of the network infrastructure starting from a library of nodes and channels (Network synthesis)
  - Communication Aware Specification and Synthesis Environment ([CASSE](#)), [FDL 2010]
  - COmmunication Synthesis Infrastructure framework ([COSI](#)), [[IEEE TASE '12](#)]
- **Open issue** : Both approaches do not rely on a standard representation of requirements (from the initial user specification) and solutions

12



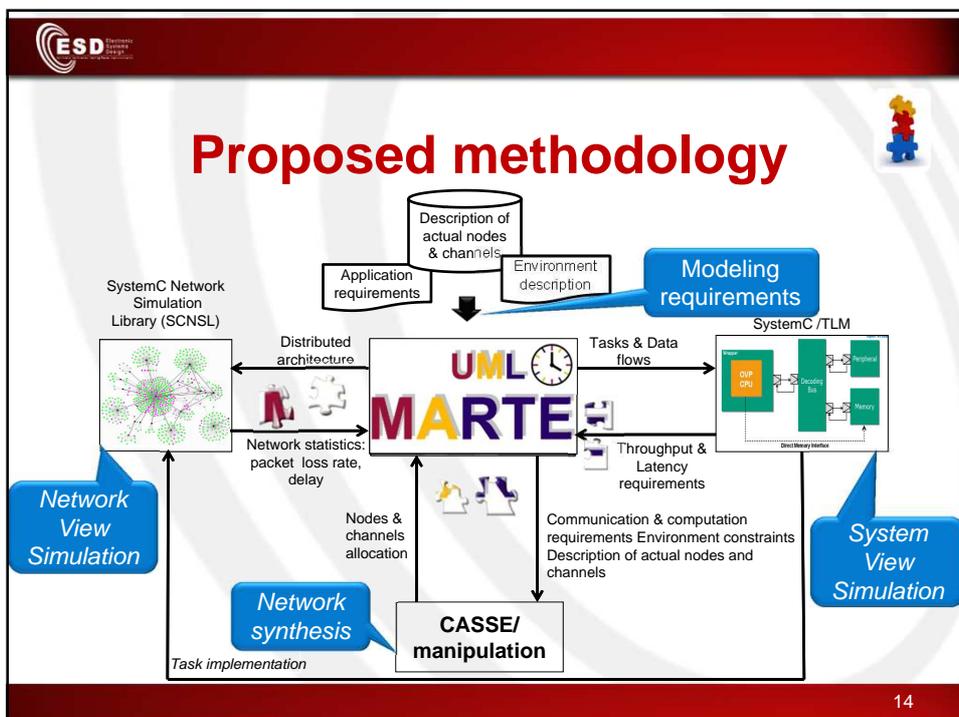
# Key idea

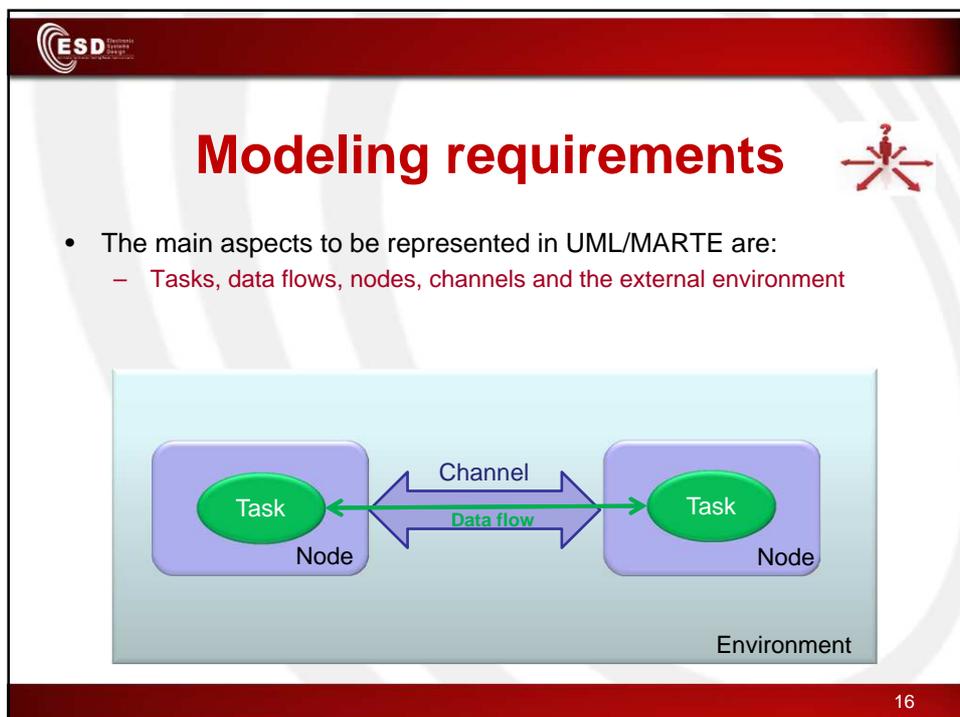
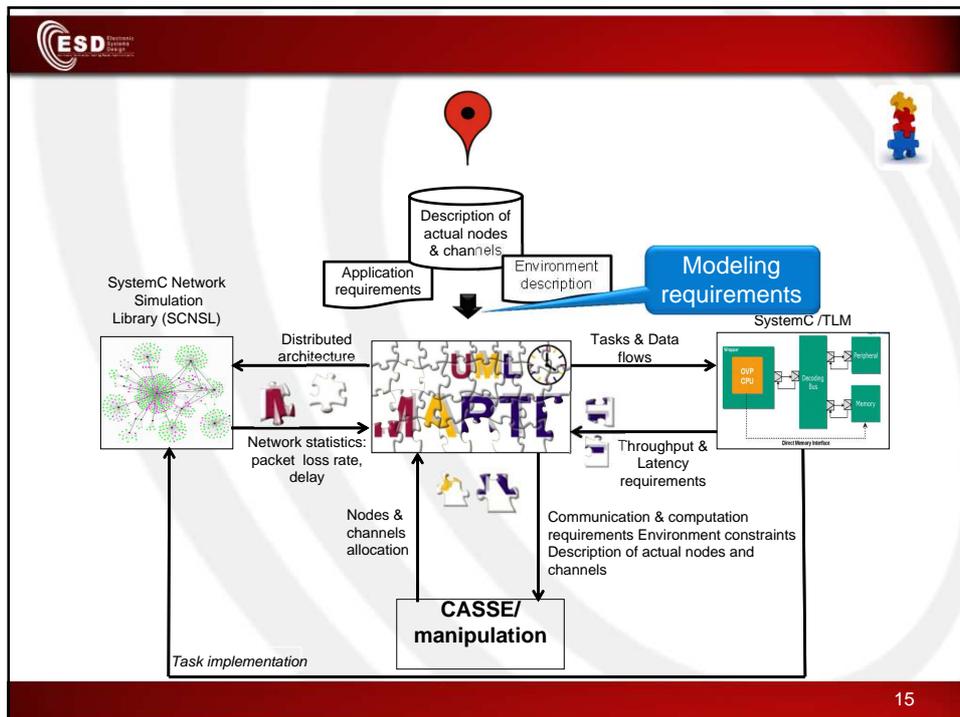


**Design methodology for networked embedded systems which combines UML/MARTE, network synthesis, and simulation**

**UML/MARTE not only at the starting point but also at the center of design flow as repository of refined version of the system up to the final solution**

13





**Modeling requirements**

- Generic Quantitative Analysis Modeling (GQAM) sub-profile of MARTE profile are used to specify the semantics of some classes and their attributes

This is the first time that GQAM is used to model the network

The diagram shows several classes and their relationships:

- Zone**: Attributes include Env\_Condition [1], LengthUnitKind [1], and n: Node [1..\*].
- Node** (Generalization of ComputationAttr): Attributes include gamma: NFP\_Real [2], k: NFP\_Price [1], m: NFP\_Boolean [1], p: NFP\_Power [1], and t: Task [1..\*].
- Task** (Generalization of ComputationAttr): Attributes include c: ComputationAttr [1] and m: NFP\_Boolean [1].
- ComputationAttr** (Generalization of Node): Attributes include mem\_size: NFP\_DataSize [1] and CPU: NFP\_Real [1].
- CommunicationAttr** (Generalization of Node): Attributes include max\_throughput: NFP\_Frequency [1], max\_delay: NFP\_Duration [1], and max\_error\_rate: NFP\_Frequency [1].
- AbstractChannel** (Generalization of Node): Attributes include d: LengthUnitKind [1], k: NFP\_Price [1], w: NFP\_Boolean [1], and n: Node [1..\*].
- DataFlow** (Generalization of Node): Attributes include ts: Task [1], td: Task [1], and c: CommunicationAttr [1].

Two magnifying glasses highlight GQAM annotations:

- Left magnifying glass** highlights the **ComputationAttr** class and its **«GaExecHost»** generalization. The annotation specifies:
 

```
memSize = ComputationAttr::mem_size
utilization = [ComputationAttr::CPU]
```
- Right magnifying glass** highlights the **CommunicationAttr** class and its **«GaCommHost»** generalization. The annotation specifies:
 

```
throughput = [CommunicationAttr::max_throughput]
blockT = [CommunicationAttr::max_delay]
```

17

**Modeling requirements**

- Generic Quantitative Analysis Modeling (GQAM) sub-profile of MARTE profile are used to specify the semantics of some classes and their attributes
- This is the first time that GQAM is used to model the network

The diagram shows several classes and their relationships:

- Zone**: Attributes include Env\_Condition [1], LengthUnitKind [1], and n: Node [1..\*].
- Node** (Generalization of ComputationAttr): Attributes include gamma: NFP\_Real [2], k: NFP\_Price [1], m: NFP\_Boolean [1], p: NFP\_Power [1], and t: Task [1..\*].
- Task** (Generalization of ComputationAttr): Attributes include c: ComputationAttr [1] and m: NFP\_Boolean [1].
- ComputationAttr** (Generalization of Node): Attributes include mem\_size: NFP\_DataSize [1] and CPU: NFP\_Real [1].
- CommunicationAttr** (Generalization of Node): Attributes include max\_throughput: NFP\_Frequency [1], max\_delay: NFP\_Duration [1], and max\_error\_rate: NFP\_Frequency [1].
- AbstractChannel** (Generalization of Node): Attributes include d: LengthUnitKind [1], k: NFP\_Price [1], w: NFP\_Boolean [1], and n: Node [1..\*].
- DataFlow** (Generalization of Node): Attributes include ts: Task [1], td: Task [1], and c: CommunicationAttr [1].

Two magnifying glasses highlight GQAM annotations:

- Left magnifying glass** highlights the **CommunicationAttr** class and its **«GaCommHost»** generalization. The annotation specifies:
 

```
throughput = [CommunicationAttr::max_throughput]
blockT = [CommunicationAttr::max_delay]
```
- Right magnifying glass** highlights the **ComputationAttr** class and its **«GaExecHost»** generalization. The annotation specifies:
 

```
memSize = ComputationAttr::mem_size
utilization = [ComputationAttr::CPU]
```

18



# Modeling requirements



- Modeling of constraint:
  - Application constraints are specified by using cardinality on the relationships between classes
- Example of constraint: “maximum one instance of t3 can be assigned to a single node”

**Task**

c: ComputationAttr [1]

m: NFP\_Boolean [1]

+ t3  
[0..1]

assigned

+ node  
[1]

**Node**

c: ComputationAttr [1]

gamma: NFP\_Real [2]

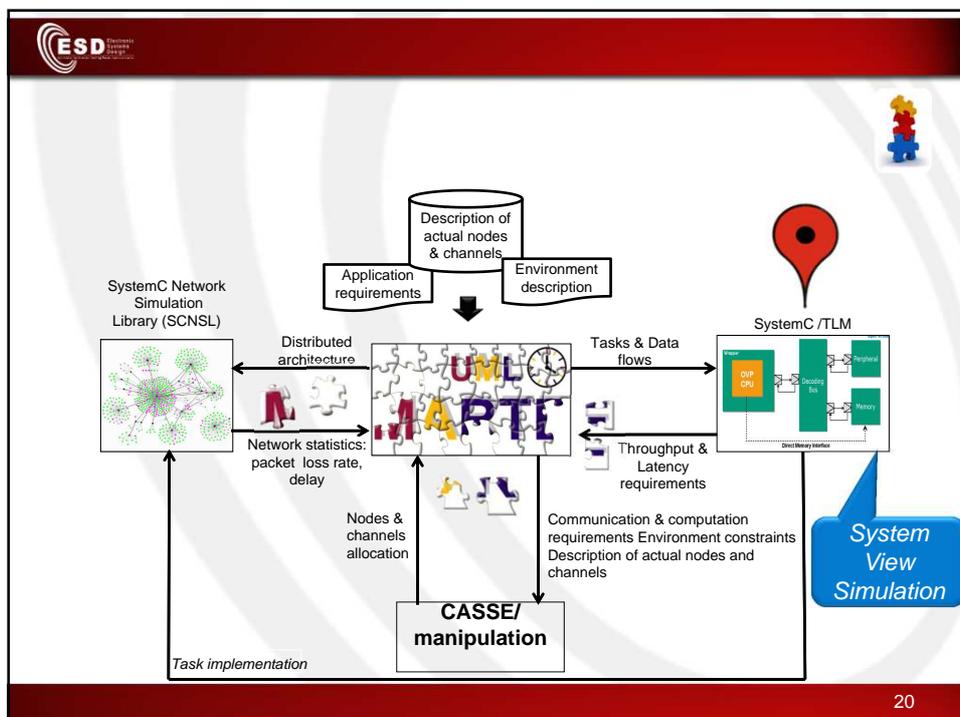
k: NFP\_Price [1]

m: NFP\_Boolean [1]

p: NFP\_Power [1]

t: Task [1..\*]

19





# System view simulation

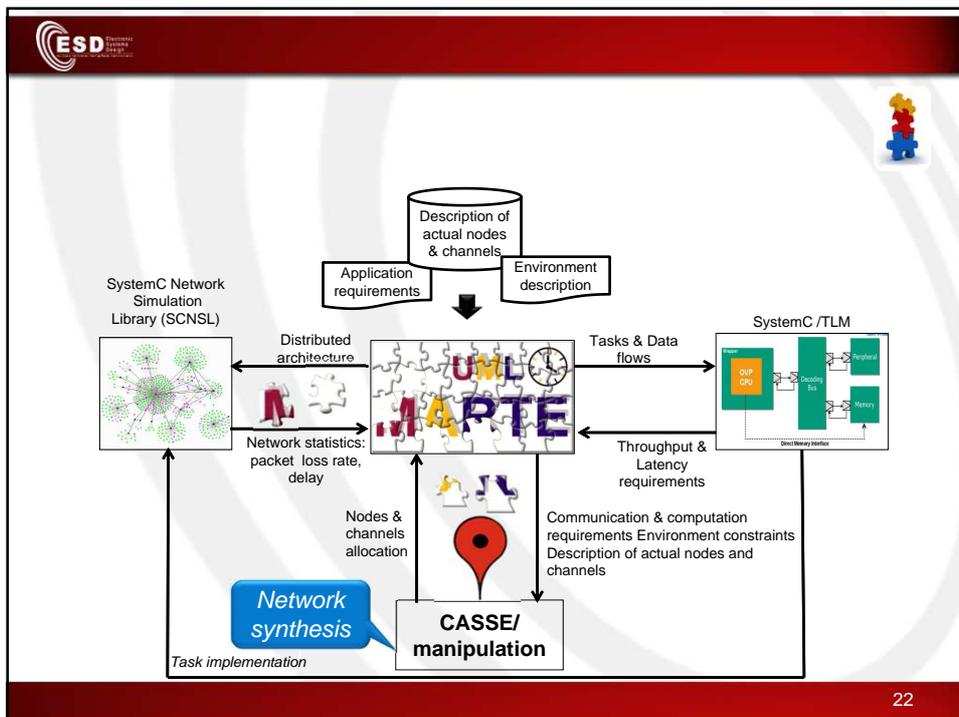


- UML/MARTE class diagram is extracted and used to generate SystemC/TLM model
  - Transformations are straight forward also (Villar,2009 and Vanderperren,2008)
- Execution of the SystemC model
  - Validate of functional behavior of the application
  - Fine-tune implementation details such as the content of exchanged messages and their sending rates
- Back annotation of throughput, latency and max error rate inside UML/MARTE model

```

classDiagram
    class DataFlow {
        ts = t3
        td = t4
    }
    class CommunicationAttr {
        max_throughput = 3
        max_delay = 1
        max_error_rate = 0.3
    }
    DataFlow -- CommunicationAttr
            
```

21





# Network synthesis



- All the information about user constraints, communication requirements and actual channels and nodes are extracted from the UML/MARTE model and translated into Network synthesis mathematical representation
- CASSE provides a mathematical notation to specify the network dimension of a distributed embedded system, preparing the way for network synthesis

**f4 : DataFlow**

ts = t3  
td = t4

---

**f4Attr : CommunicationAttr**

max\_throughput = 3  
max\_delay = 1  
max\_error\_rate = 0.3

⇒

*Dataflow(f4) = [t3, t4, [3, 1, 0.3]].*

23



# Network synthesis cont'd



Set of tasks & data flows

<b>t3 : Task</b>	<b>f4 : DataFlow</b>
c = 1, 1 m = true	ts = t3 td = t4 c = 3, 1, 0.3

UML deployment diagram:  
Assignment of tasks inside nodes  
and data flows inside channels

The building geometry

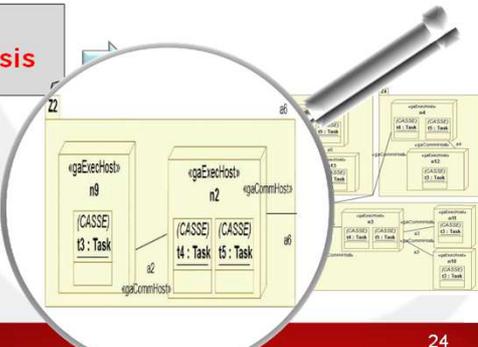
**z3 : Zone**

→

NW  
Synthesis

Technological library (network nodes and channels)

<b>b : Node</b>	<b>y : AbstractChannel</b>
c = 10, 10 gamma = [0.6, 0.9] m = false k = 100	c = 54.2, 0.1 d = 30 w = true k = 20



24

**ESD** Elements System Design

## Manipulation

- This step aims at obtaining several NW alternatives which are equivalent from the network perspective
- Mathematical-based rules
  - Divide
  - Split
  - Merge
  - Aggregate

25

**ESD** Elements System Design

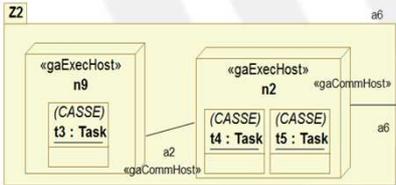
26



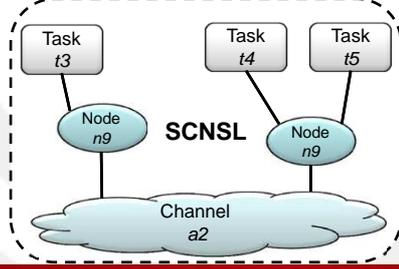

## Network view simulation

- **SCNSL** is an extension of SystemC to allow modeling packet-based networks
  - It allows the easy and complete modeling of distributed applications of networked embedded systems such as wireless sensor networks, routers, and distributed plant controllers

Z2



UML deployment diagram



SCNSL

27



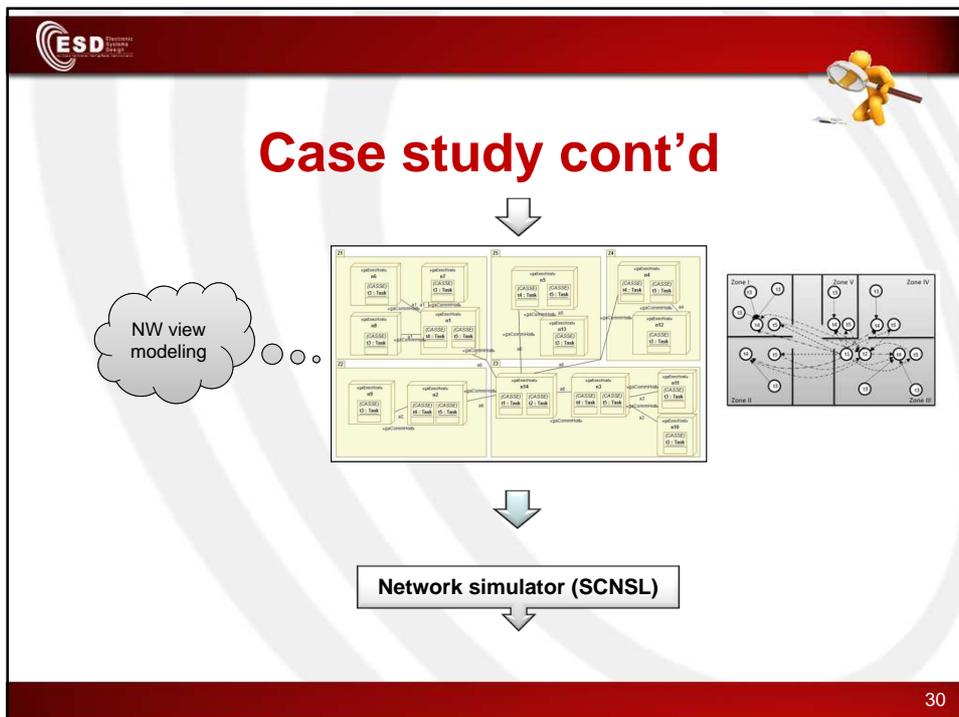
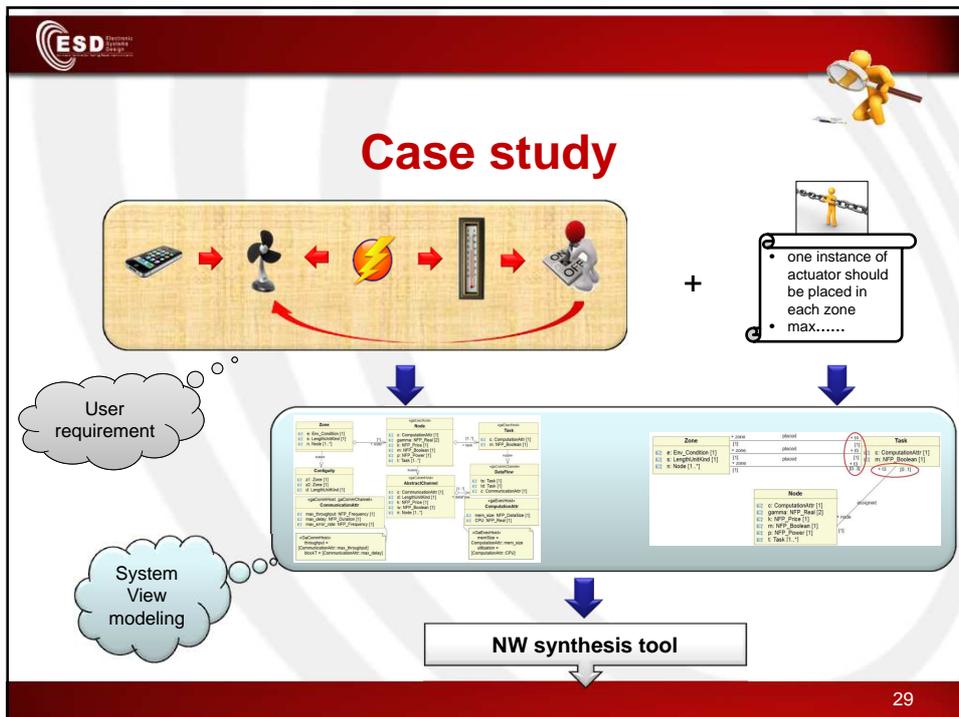

## Network view simulation

- Correspondence between UML/MARTE and SCNSL elements

UML/MARTE	SCNSL
Node (n1)	<code>n1 = scnsl-&gt;createNode();</code>
Channel (ch) bound to node (n1)	<code>CoreChannelSetup t ccs; ch = scnsl-&gt;createChannel(ccs); BindSetup base t bsb1; scnsl-&gt;bind(n1,ch,bsb1);</code>
Data flow between task (t1) and task (t2)	<code>CoreCommunicatorSetup t ccoms; mac1 = scnsl-&gt;createCommunicator(ccoms); scnsl-&gt;bind(&amp;t1, &amp;t2, ch, bsb1, mac1);</code>

E. Ebeid, F. Fummi, D. Quaglia, F. Stefanni

28

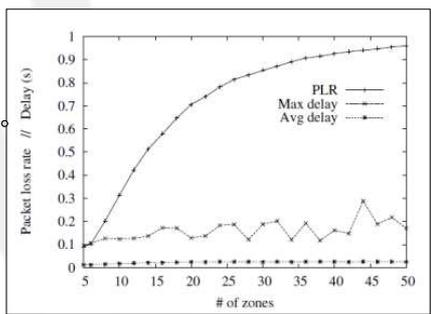





## Case study cont'd



NW simulation statistics



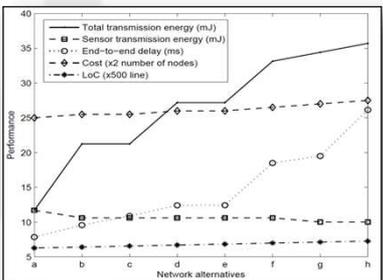
31

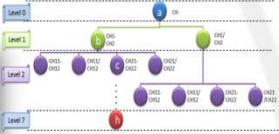



## Case study cont'd



NW simulation statistics





NW manipulation

32



## Summarization

- User requirements and constraints has been modeled by using UML/MARTE profile and simulated by SystemC/TLM at system view level
- Simulation results has been used to refine the user model
- Network synthesis tools have been used to solve the application problem
- Network solutions have been modeled and simulated by using SCNSL
- Network statistics have been used for the final refinement of application model
- **M**anipulation and **A**utomatic design-space exploration

33



## Conclusions

- Some UML/MARTE diagrams and stereotypes have been used as a first time to represent the building blocks of a distributed embedded application
  - Elements from the MARTE specification have been applied to the context of distributed embedded applications
- Some gaps in MARTE standard have been identified concerning the representation of constraints and attributes related to error rate information
- SystemC code has been generated for both functional and network-aware simulation

**A UML-centric design flow for networked embedded systems has been created**

34