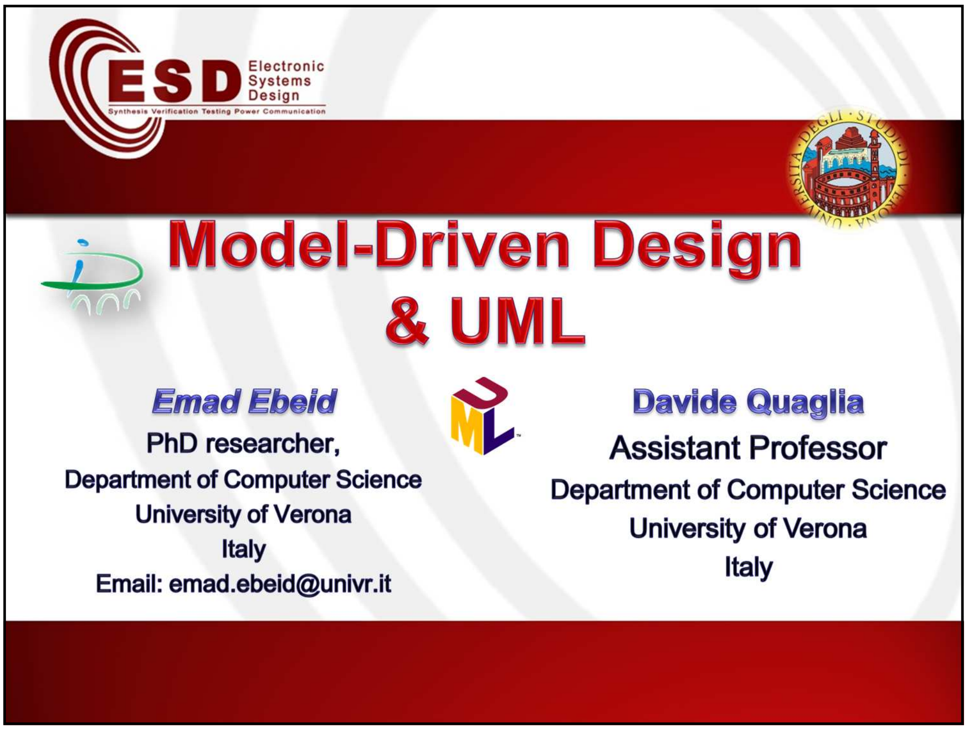


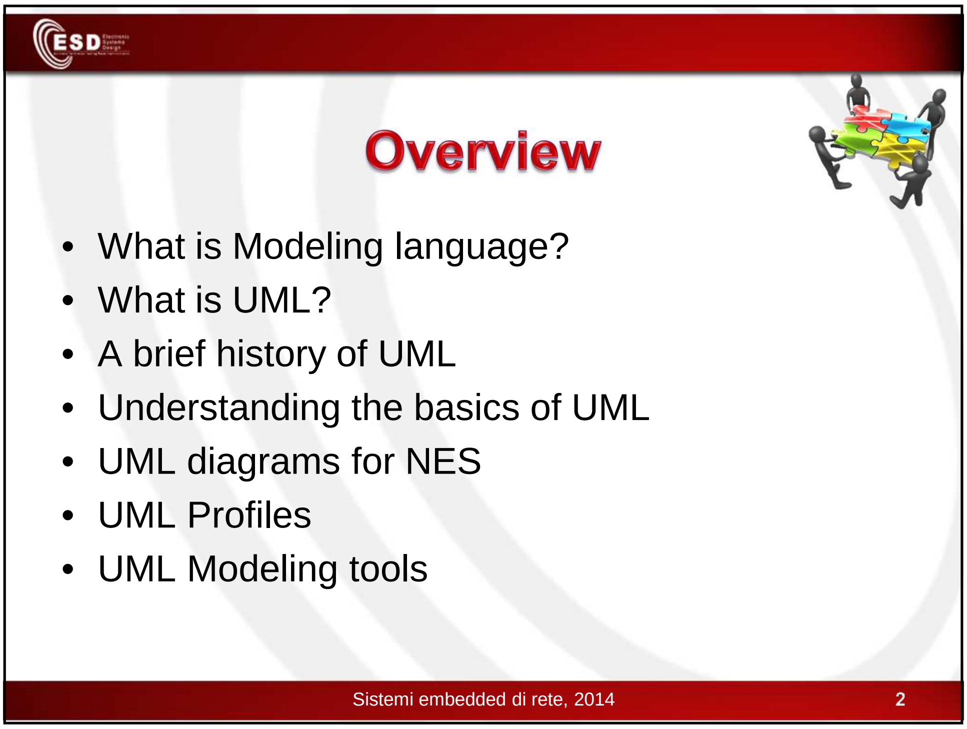
ESD Electronic Systems Design

Synthesis Verification Testing Power Communication

Model-Driven Design & UML

Emad Ebeid
PhD researcher,
Department of Computer Science
University of Verona
Italy
Email: emad.ebeid@univr.it

Davide Quaglia
Assistant Professor
Department of Computer Science
University of Verona
Italy



Overview

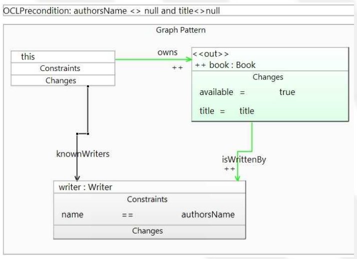
- What is Modeling language?
- What is UML?
- A brief history of UML
- Understanding the basics of UML
- UML diagrams for NES
- UML Profiles
- UML Modeling tools

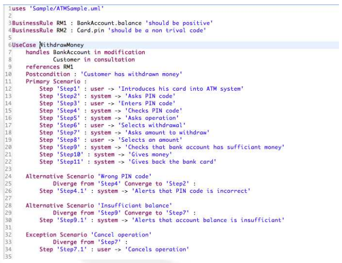
Sistemi embedded di rete, 2014 2

What is Modeling language?

A modeling language is any artificial language that can be used to express *information*, *knowledge* or *systems* in a structure that is defined by a consistent set of rules. The rules are used for interpretation of the meaning of components in the structure

- A modeling language can be graphical or textual

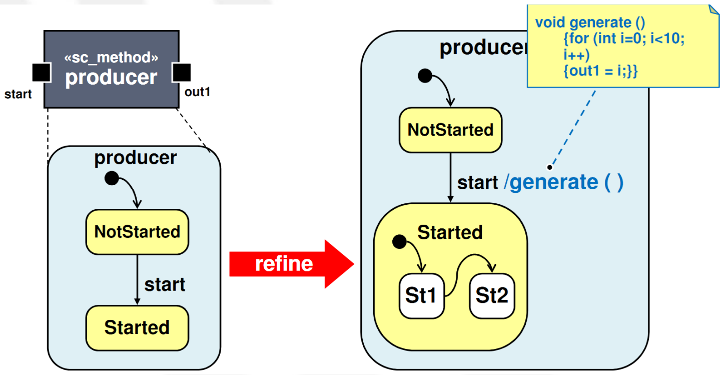




```
uses 'Sample/ATMSample.uml'

BusinessRule RM1 : BankAccount.balance 'should be positive'
BusinessRule RM2 : Card.pin 'should be a non trival code'

UseCase WithdrawMoney
    handles BankAccount in modification
            Customer in consultation
    references RM1
    Postcondition : 'Customer has withdrawn money'
    Primary Scenario :
        Step 'Step1' : user -> 'Introduces his card into ATM system'
        Step 'Step2' : system -> 'Asks PIN code'
        Step 'Step3' : user -> 'Enters PIN code'
        Step 'Step4' : system -> 'Checks PIN code'
        Step 'Step5' : system -> 'Asks operation'
        Step 'Step6' : user -> 'Selects withdrawal'
        Step 'Step7' : system -> 'Asks amount to withdraw'
        Step 'Step8' : user -> 'Selects an amount'
        Step 'Step9' : system -> 'Checks that bank account has sufficiant money'
        Step 'Step10' : system -> 'Gives money'
        Step 'Step11' : system -> 'Gives back the bank card'

    Alternative Scenario 'Wrong PIN code'
            Diverge from 'Step4' Converge to 'Step2' :
        Step 'Step4.1' : system -> 'Alerts that PIN code is incorrect'

    Alternative Scenario 'Insufficiant balance'
            Diverge from 'Step9' Converge to 'Step7' :
        Step 'Step9.1' : system -> 'Alerts that account balance is insufficiant'

    Exception Scenario 'Cancel operation'
            Diverge from 'Step7' :
        Step 'Step7.1' : user -> 'Cancels operation'

```

Model-based development

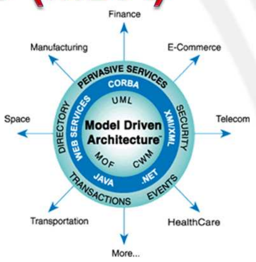
- Models can be refined continuously until the application is fully specified



```
void generate ()
    {for (int i=0; i<10;
    i++)
    {out1 = i;}}
```

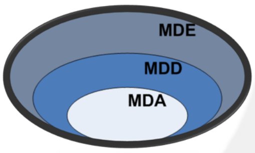
Model-Driven Architecture (MDA)™

- It was launched by the Object Management Group (OMG) in 2001
- MDA provide portability, interoperability, maintainability and reusability of models
- MDA approach defines system functionality using a platform-independent model (PIM) using an **appropriate domain-specific language**



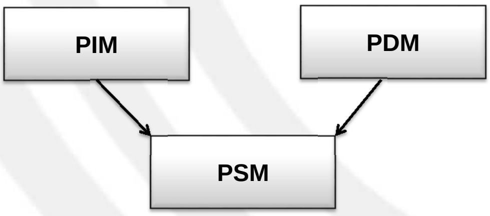
Model-driven development (MDD) is a software engineering *approach* that uses model to create a product. It may refer to specific *tools* and resources.

Model-driven engineering (**MDE**) is a software development *methodology* which focuses on creating and exploiting domain models



Model-Driven Architecture viewpoints

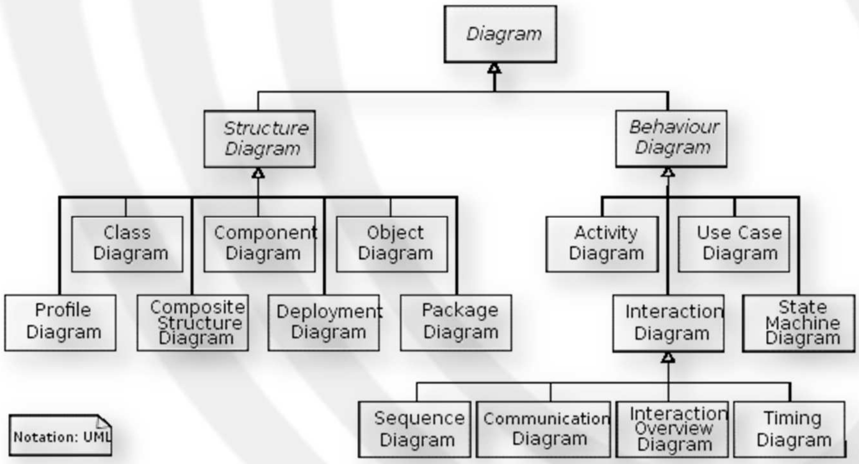
- **The Platform Independent Model (PIM):** The functional and non-functional aspects
- **The Platform Description Model (PDM):** HW and SW resources
- **The Platform Specific Model (PSM):** System architecture



What is UML?

- **Unified Modeling Language (UML)** is a standardized general-purpose modeling language in the field of object-oriented software engineering
  - UML can be applied in many areas like embedded systems, web applications, commercial applications etc.
- The standard was created, and is managed by the Object Management Group (OMG)

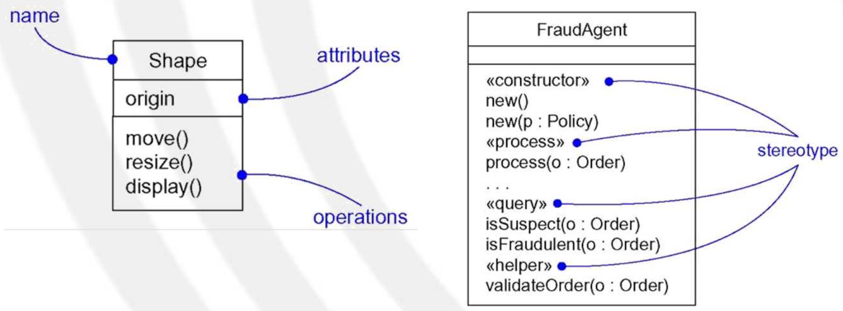
UML diagrams



Why UML for Modeling

- Use graphical notation to communicate more clearly than natural language (imprecise) and less detailed than code
- Help to acquire an overall view of a system
- UML is *not* dependent on any one language or technology
- UML moves us from fragmentation to standardization

Class Diagram



name

Shape

origin

move()
resize()
display()

attributes

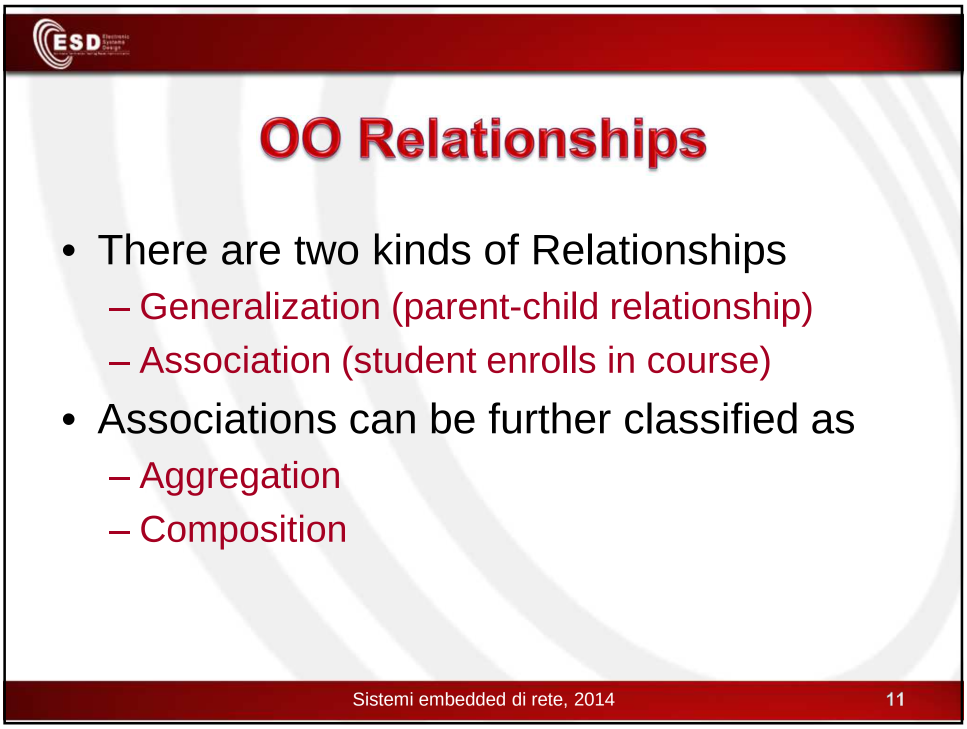
operations

FraudAgent

«constructor»
new()
new(p : Policy)
«process»
process(o : Order)
. . .
«query»
isSuspect(o : Order)
isFraudulent(o : Order)
«helper»
validateOrder(o : Order)

stereotype

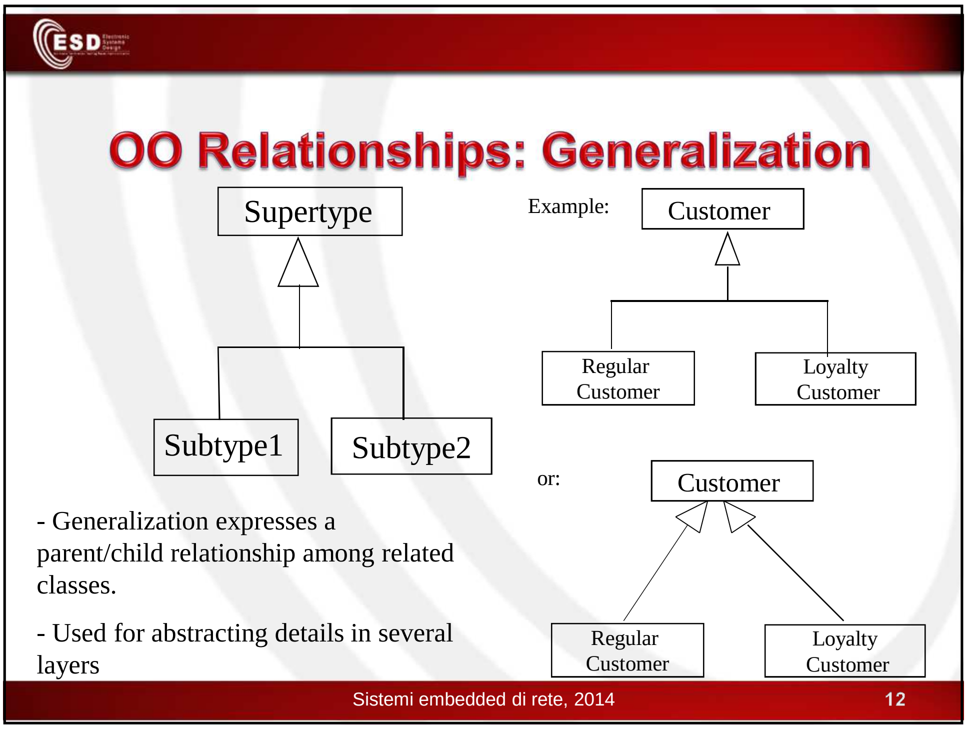
Used to specify Object Oriented (OO) paradigm



OO Relationships

- There are two kinds of Relationships
  - Generalization (parent-child relationship)
  - Association (student enrolls in course)
- Associations can be further classified as
  - Aggregation
  - Composition

Sistemi embedded di rete, 2014 11



OO Relationships: Generalization

- Generalization expresses a parent/child relationship among related classes.

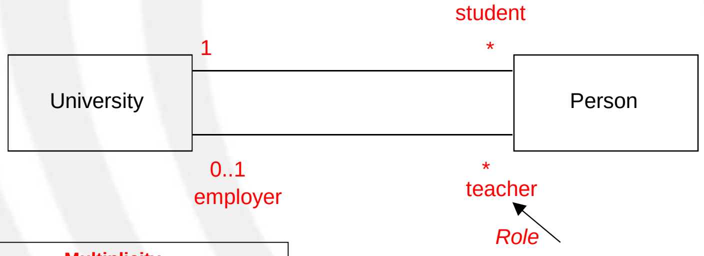
- Used for abstracting details in several layers

Sistemi embedded di rete, 2014 12

OO Relationships: Association

- Represent relationship between instances of classes
  - Student enrolls in a course
  - Courses have students
  - Courses have exams
  - Etc.
- Association has two ends
  - Role names (e.g. enrolls)
  - Multiplicity (e.g. One course can have many students)

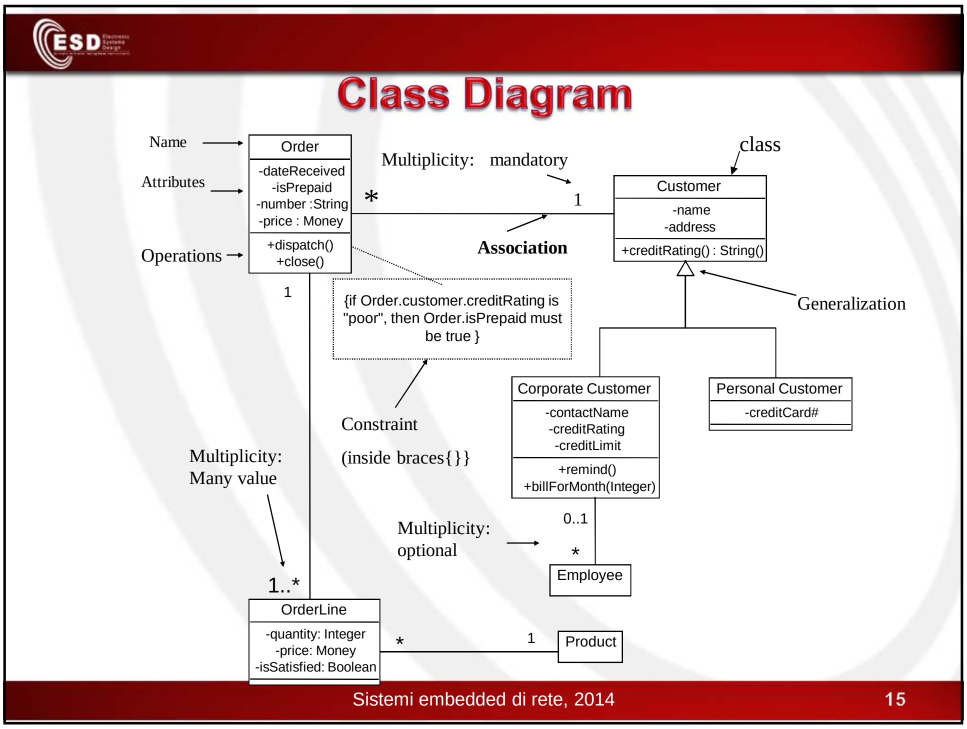
Association: Multiplicity and Roles



Multiplicity	
Symbol	Meaning
1	One and only one
0..1	Zero or one
M..N	From M to N (natural language)
*	From zero to any positive integer
0..*	From zero to any positive integer
1..*	From one to any positive integer

Role

"A given university groups many people; some act as students, others as teachers. A given student belongs to a single university; a given teacher may or may not be working for the university at a particular time."



Class Diagram

Name

Attributes

Operations

Order

-dateReceived
-isPrepaid
-number :String
-price : Money

+dispatch()
+close()

Multiplicity: mandatory

*

1

Association

class

Customer

-name
-address

+creditRating() : String()

Generalization

{if Order.customer.creditRating is "poor", then Order.isPrepaid must be true }

Constraint

(inside braces{}}

1

Multiplicity: Many value

1..*

Corporate Customer

-contactName
-creditRating
-creditLimit

+remind()
+billForMonth(Integer)

Personal Customer

-creditCard#

Multiplicity: optional

0..1

*

Employee

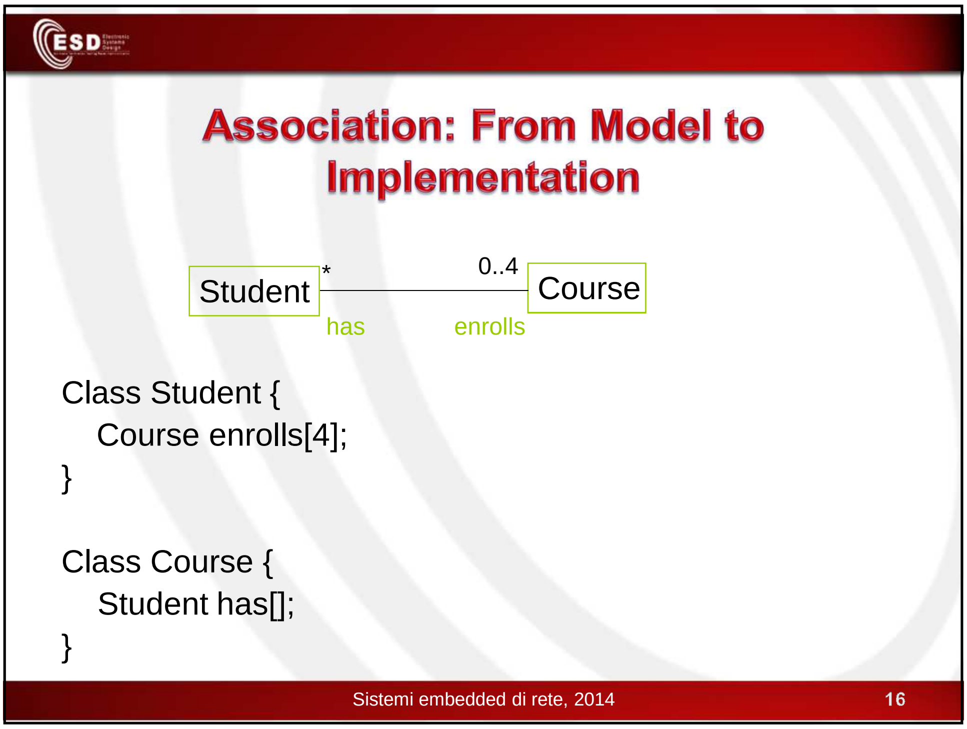
OrderLine

-quantity: Integer
-price: Money
-isSatisfied: Boolean

*

1

Product

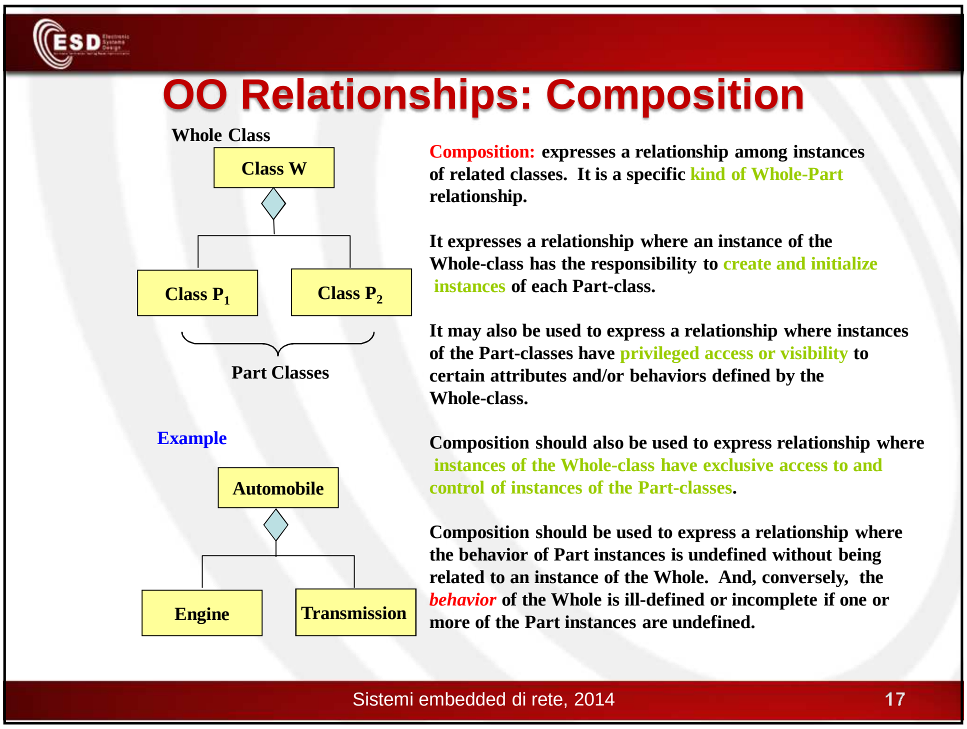


Association: From Model to Implementation

Student * 0..4 Course

has enrolls

```
Class Student {
   Course enrolls[4];
}

Class Course {
   Student has[];
}
```



OO Relationships: Composition

Whole Class

Class W

Class P_1

Class P_2

Part Classes

Example

Automobile

Engine

Transmission

Composition: expresses a relationship among instances of related classes. It is a specific kind of Whole-Part relationship.

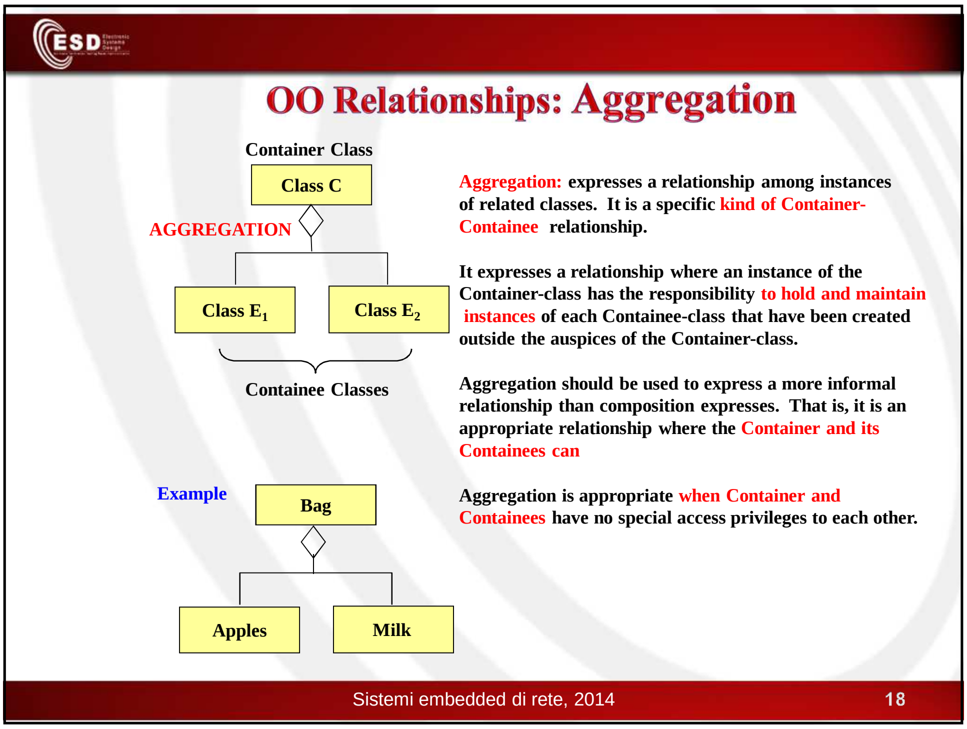
It expresses a relationship where an instance of the Whole-class has the responsibility to create and initialize instances of each Part-class.

It may also be used to express a relationship where instances of the Part-classes have privileged access or visibility to certain attributes and/or behaviors defined by the Whole-class.

Composition should also be used to express relationship where instances of the Whole-class have exclusive access to and control of instances of the Part-classes.

Composition should be used to express a relationship where the behavior of Part instances is undefined without being related to an instance of the Whole. And, conversely, the *behavior* of the Whole is ill-defined or incomplete if one or more of the Part instances are undefined.

Sistemi embedded di rete, 2014 17



OO Relationships: Aggregation

Container Class

Class C

AGGREGATION

Class E_1

Class E_2

Containee Classes

Example

Bag

Apples

Milk

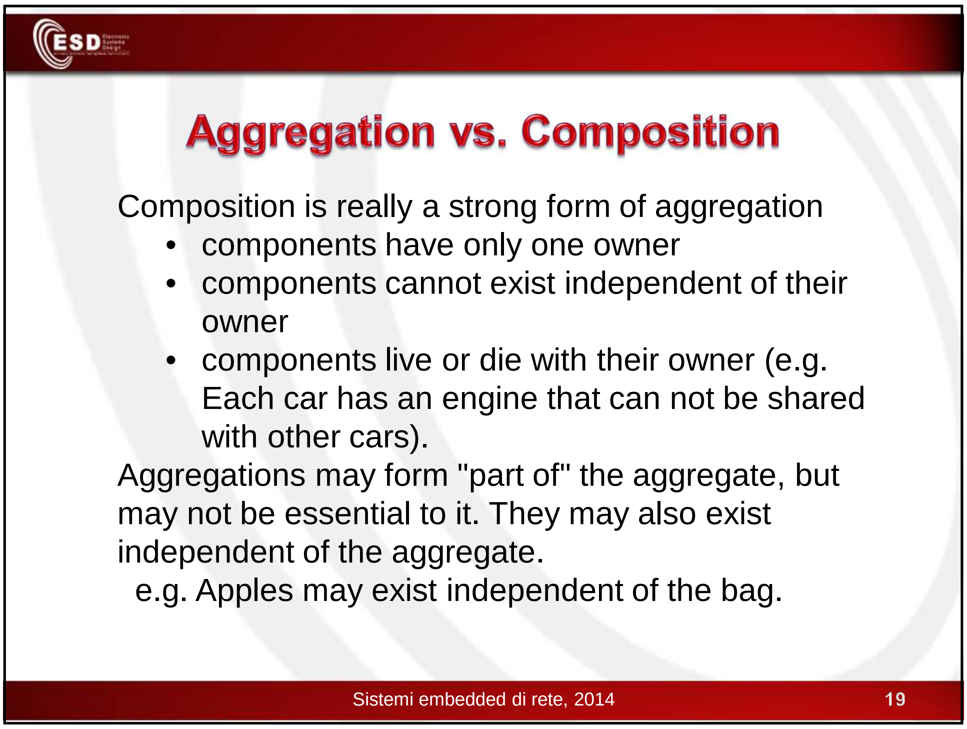
Aggregation: expresses a relationship among instances of related classes. It is a specific kind of Container-Containee relationship.

It expresses a relationship where an instance of the Container-class has the responsibility to hold and maintain instances of each Containee-class that have been created outside the auspices of the Container-class.

Aggregation should be used to express a more informal relationship than composition expresses. That is, it is an appropriate relationship where the Container and its Containees can

Aggregation is appropriate when Container and Containees have no special access privileges to each other.

Sistemi embedded di rete, 2014 18



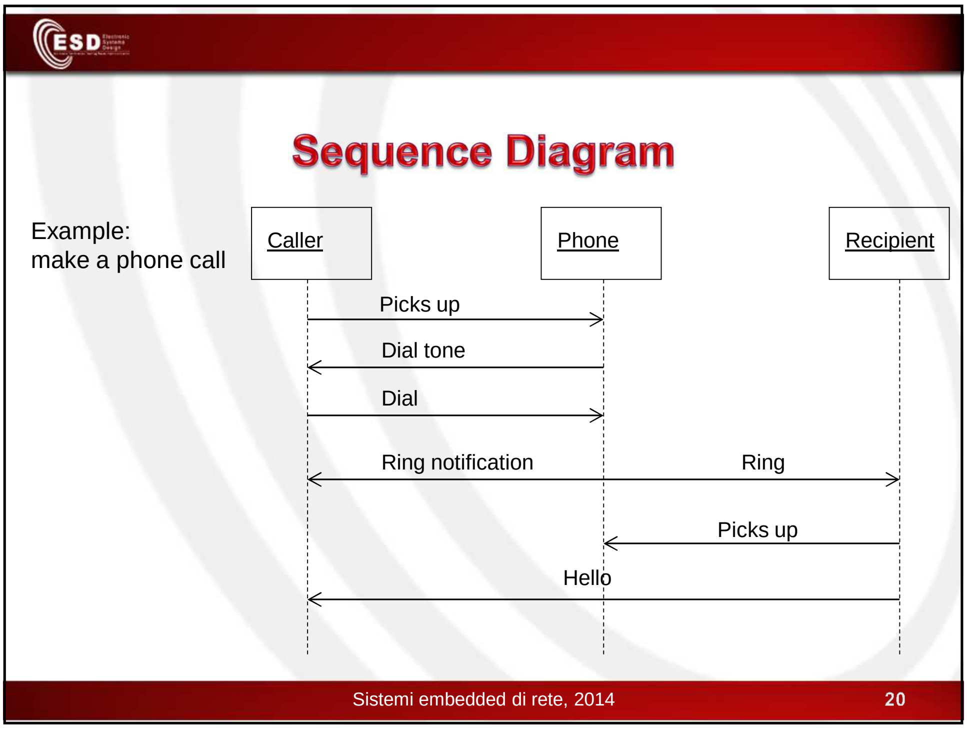
Aggregation vs. Composition

Composition is really a strong form of aggregation

- components have only one owner
- components cannot exist independent of their owner
- components live or die with their owner (e.g. Each car has an engine that can not be shared with other cars).

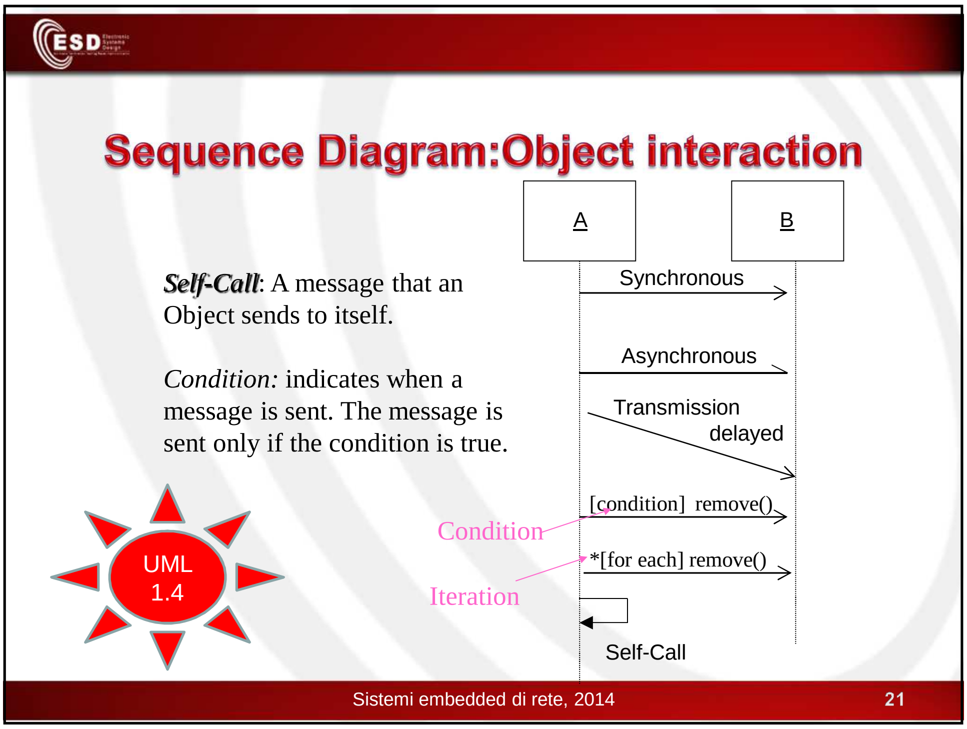
Aggregations may form "part of" the aggregate, but may not be essential to it. They may also exist independent of the aggregate.

e.g. Apples may exist independent of the bag.



Sequence Diagram

Example:
make a phone call

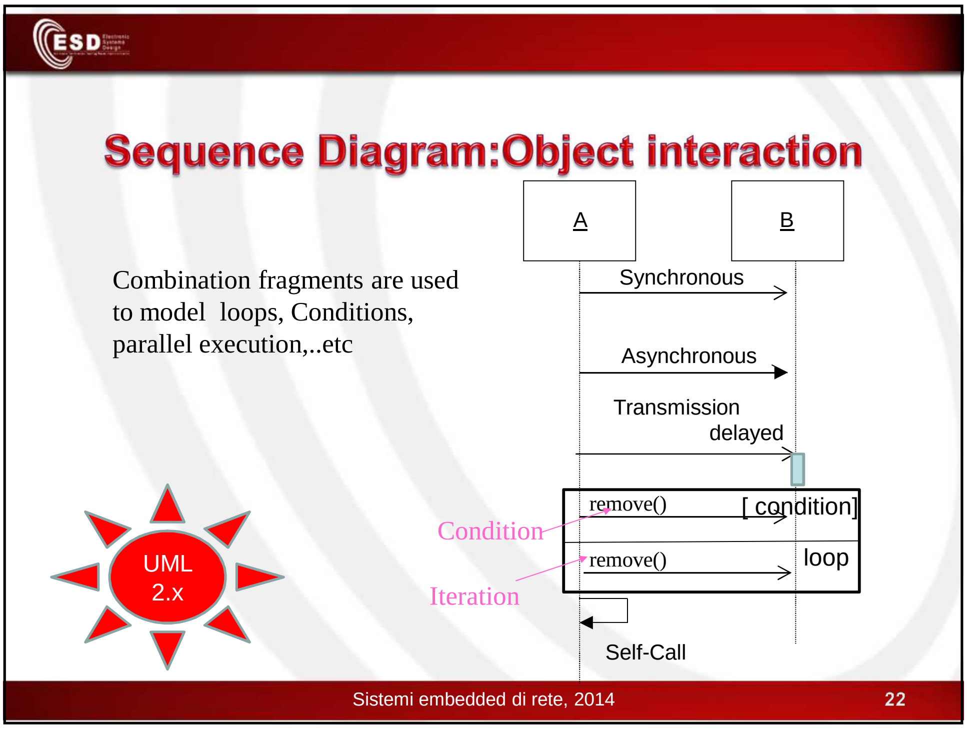


Sequence Diagram:Object interaction

Self-Call: A message that an Object sends to itself.

Condition: indicates when a message is sent. The message is sent only if the condition is true.

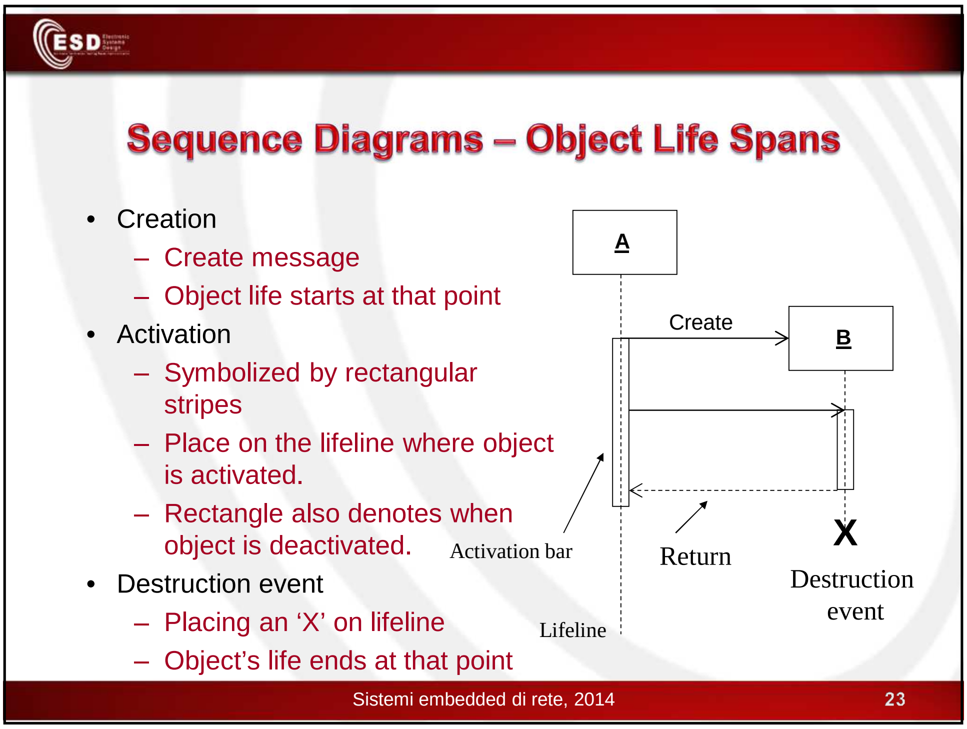
Sistemi embedded di rete, 2014 21



Sequence Diagram:Object interaction

Combination fragments are used to model loops, Conditions, parallel execution,..etc

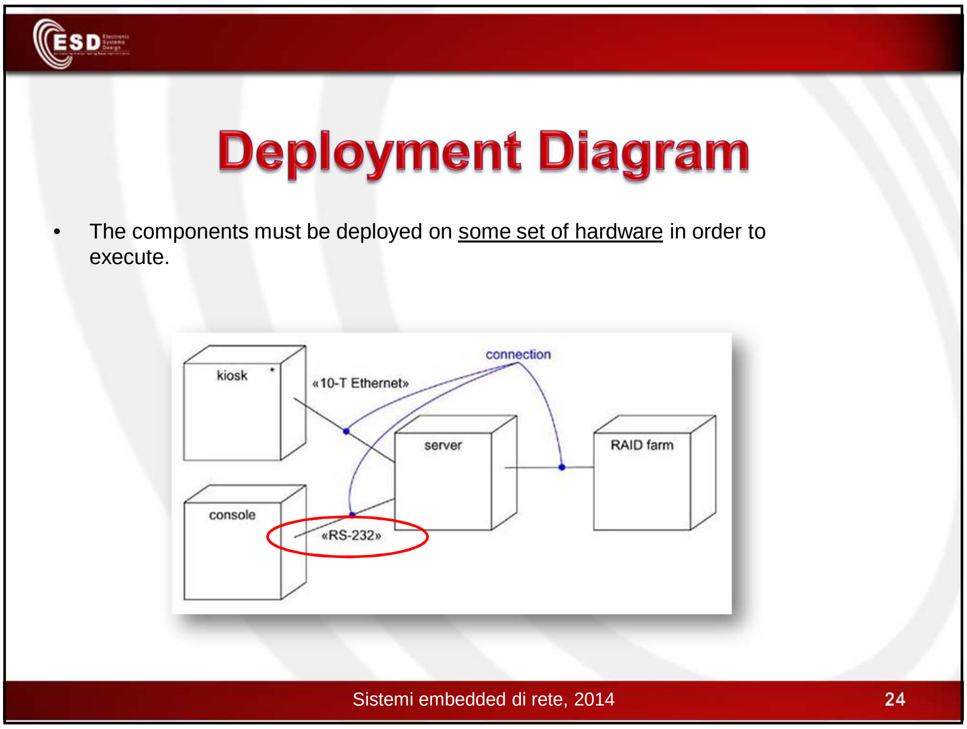
Sistemi embedded di rete, 2014 22



Sequence Diagrams – Object Life Spans

- Creation
  - Create message
  - Object life starts at that point
- Activation
  - Symbolized by rectangular stripes
  - Place on the lifeline where object is activated.
  - Rectangle also denotes when object is deactivated.
- Destruction event
  - Placing an ‘X’ on lifeline
  - Object’s life ends at that point

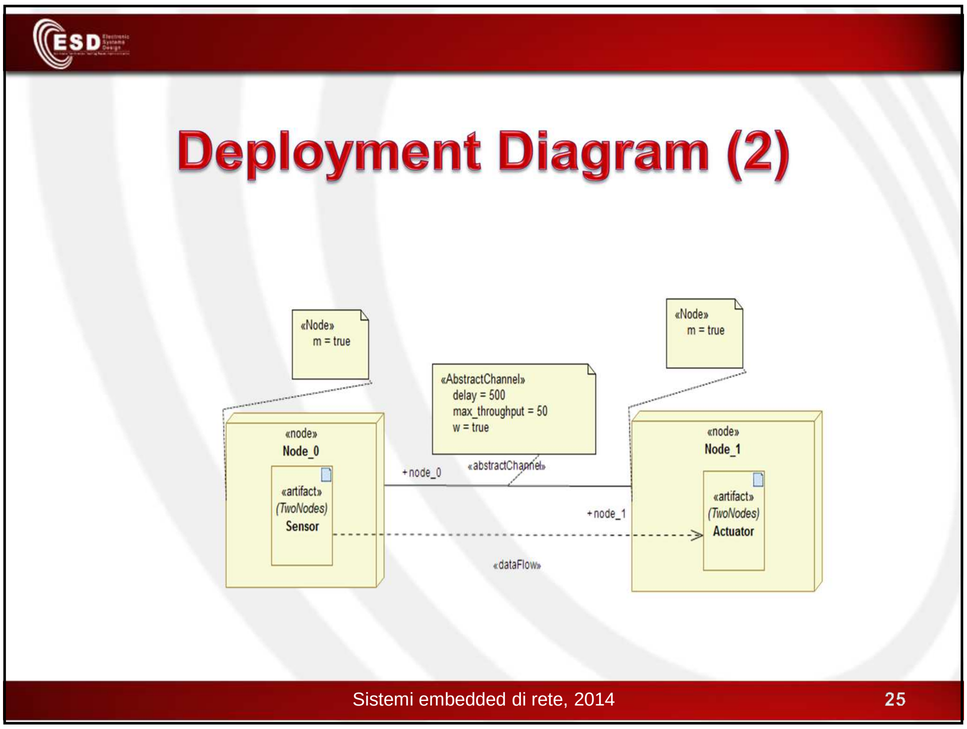
Sistemi embedded di rete, 2014 23



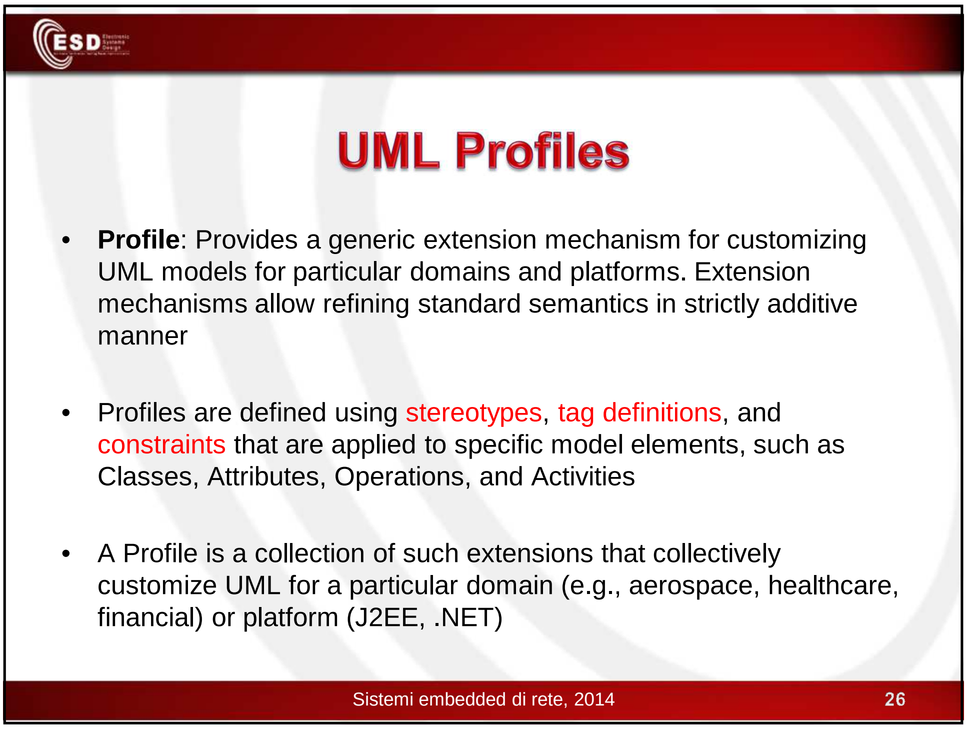
Deployment Diagram

- The components must be deployed on some set of hardware in order to execute.

Sistemi embedded di rete, 2014 24



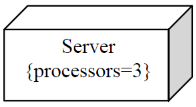
Deployment Diagram (2)



UML Profiles

- **Profile**: Provides a generic extension mechanism for customizing UML models for particular domains and platforms. Extension mechanisms allow refining standard semantics in strictly additive manner
- Profiles are defined using stereotypes, tag definitions, and constraints that are applied to specific model elements, such as Classes, Attributes, Operations, and Activities
- A Profile is a collection of such extensions that collectively customize UML for a particular domain (e.g., aerospace, healthcare, financial) or platform (J2EE, .NET)

Tagged Values



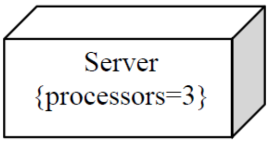
A tagged value is a combination of a tag and a value that gives supplementary information that is attached to a model element. A tagged value can be used to add properties to any model elements and can be applied to a model element or a stereotype.

Tagged values can be defined for existing model elements, or for individual stereotypes, so that everything with that stereotype has that tagged value. It is important to mention that a tagged value is not equal to a class attribute. Instead, you can regard a tagged value as being a metadata, since its value applies to the element itself and not to its instances.

One of the most common uses of a tagged value is to *specify properties* that are relevant to code generation or configuration management. So, for example, you can make use of a tagged value in order to specify the programming language to which you map a particular class, or you can use it to denote the author and the version of a component.

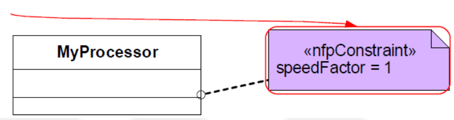
Tagged Values

- Graphically, a tagged value is rendered as a string enclosed by brackets, which is placed below the name of another model element. The string consists of a name (the tag), a separator (the symbol =), and a value (of the tag)



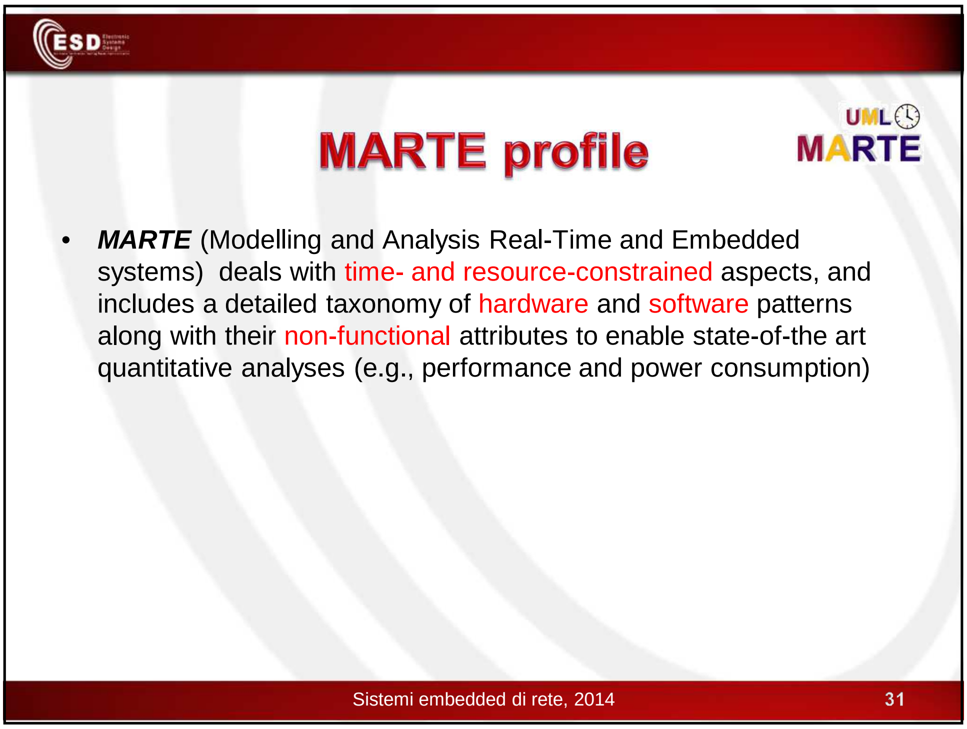
Constraints

- Constraints are properties for specifying semantics and/or conditions that must be held true at all times for the elements of a model. They allow you to extend the semantics of a UML building block by adding new rules, or modifying existing ones.
- For example, when modeling hard real time systems it could be useful to adorn the models with some additional information, such as time budgets and deadlines. By making use of constraints these timing requirements can easily be captured.



Catalog of Adopted OMG Profiles

- UML Profile for CORBA
- UML Profile for Enterprise Application Integration (EAI)
- UML Profile for Enterprise Distributed Object Computing (EDOC)
- UML Profile for Modeling QoS and Fault Tolerance Characteristics and Mechanisms
- UML Profile for Schedulability, Performance, and Time
- UML Profile for System on a Chip (SoC)
- UML Profile for Modeling and Analysis of Real-Time and Embedded Systems (MARTE)
- UML Testing Profile
- UML Profile for Systems Engineering (SysML)
- UML Profile for DoDAF/MoDAF (UPDM)

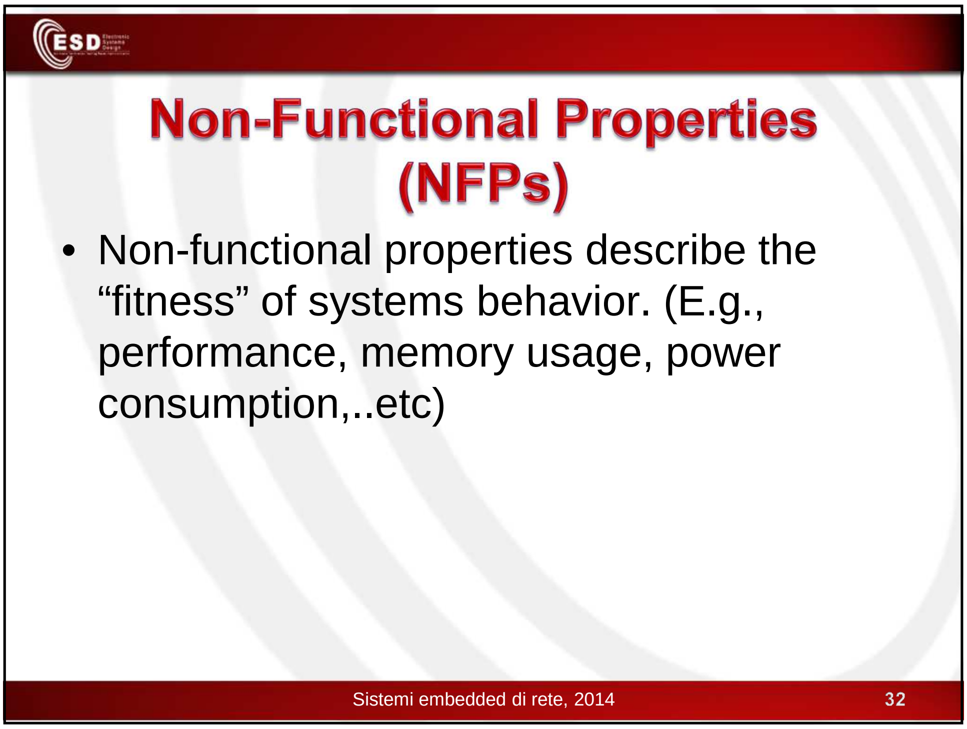


MARTE profile

UML MARTE

- *MARTE* (Modelling and Analysis Real-Time and Embedded systems) deals with time- and resource-constrained aspects, and includes a detailed taxonomy of hardware and software patterns along with their non-functional attributes to enable state-of-the art quantitative analyses (e.g., performance and power consumption)

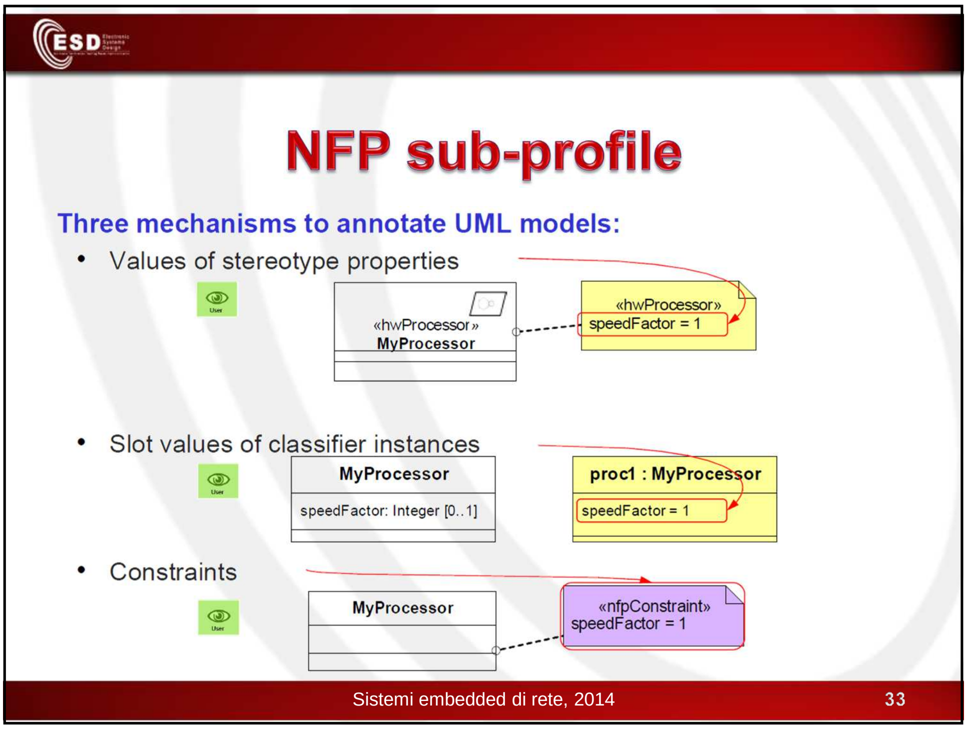
Sistemi embedded di rete, 2014 31



Non-Functional Properties (NFPs)

- Non-functional properties describe the “fitness” of systems behavior. (E.g., performance, memory usage, power consumption,..etc)

Sistemi embedded di rete, 2014 32



NFP sub-profile

Three mechanisms to annotate UML models:

- Values of stereotype properties

«hwProcessor»
MyProcessor

«hwProcessor»
speedFactor = 1

- Slot values of classifier instances

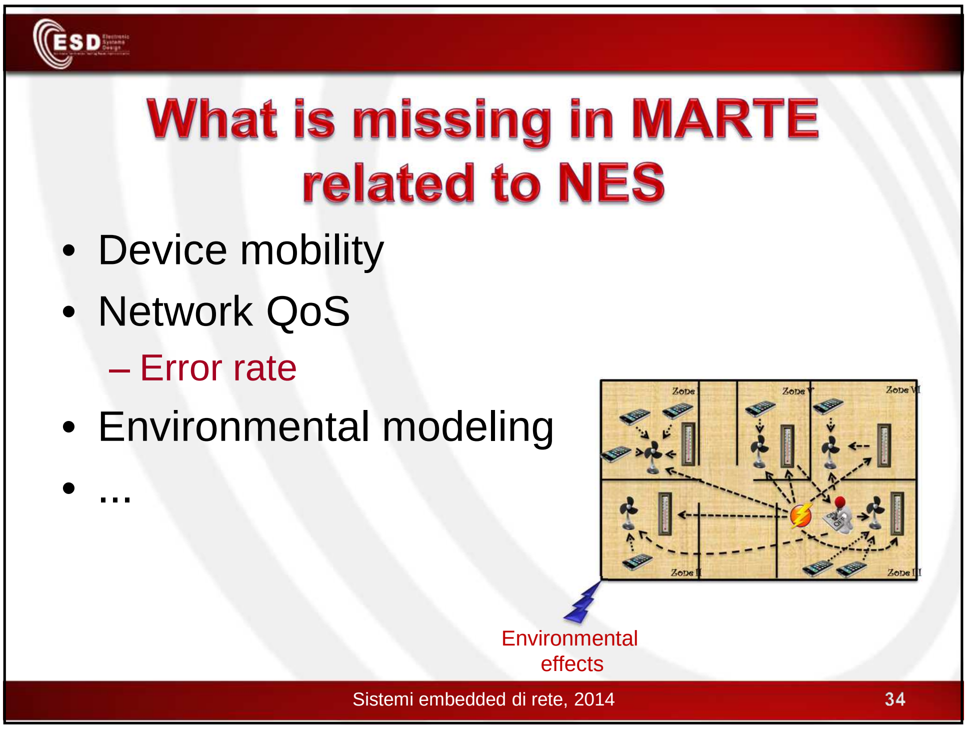
MyProcessor
speedFactor: Integer [0..1]

proc1 : MyProcessor
speedFactor = 1

- Constraints

MyProcessor

«nfpConstraint»
speedFactor = 1

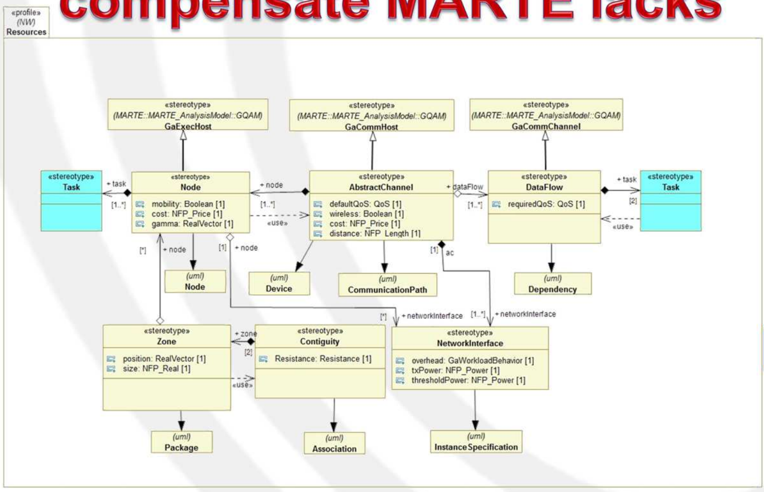


What is missing in MARTE related to NES

- Device mobility
- Network QoS
  - Error rate
- Environmental modeling
- …

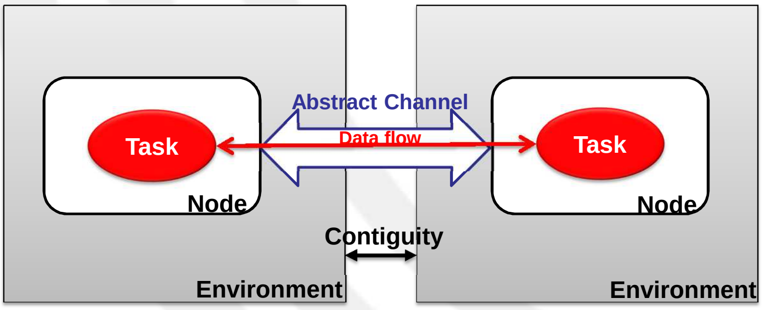
Environmental effects

UML Network Profile to compensate MARTE lacks



DSD'13

NES elements



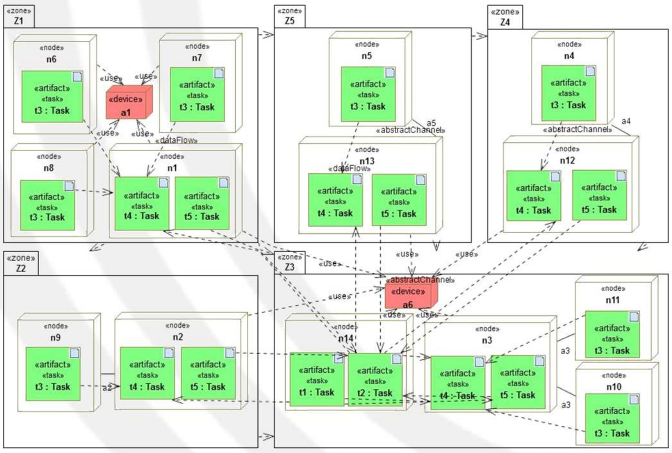
NES elements

1. **Task (t),** represents the functional part of the application which can be periodic or aperiodic and it can be data producer or consumer.
2. **Data flow (f),** represents the communication link between two entities of type task (t). f expresses the communication requirements between task such as minimum throughput, maximum delay and error rate.
3. **Node (n),** represents the physical element of the network which will host one or more tasks to be run on it. It can be a hardware component that has processing unit, memory and at least one network interface.
4. **Abstract Channel (ac),** represents the communication link between one or more entities of type node (n). Ac can be wired or wireless and it defines the characteristics of the channel (i.e., delay, capacity. error rate).
5. **Zone (z),** represents a partitioning of the physical environment in which the set of NES's is deployed. It groups nodes and defines their position. Furthermore, it captures the relevant environmental parameters such as room temperature in a temperature monitoring application.

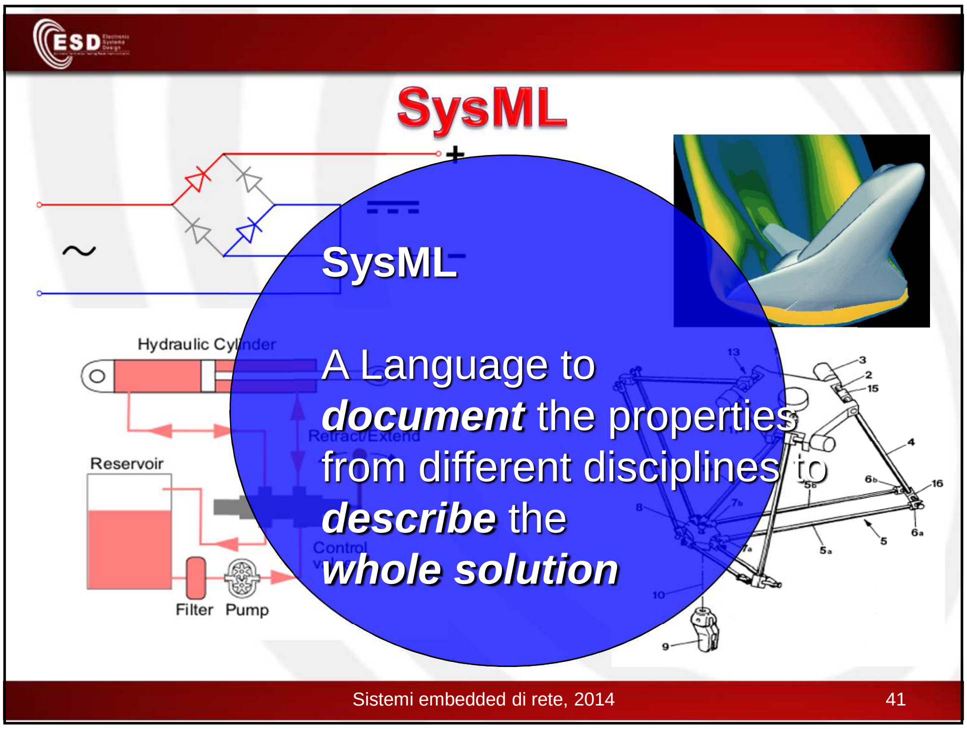
NES elements

6. **Contiguity (c),** represents the relationship between two entities of type zone (z). c captures the environmental characteristics between two zones which affect inter-zone communication

UML/Deployment diagram + NW profile for NES



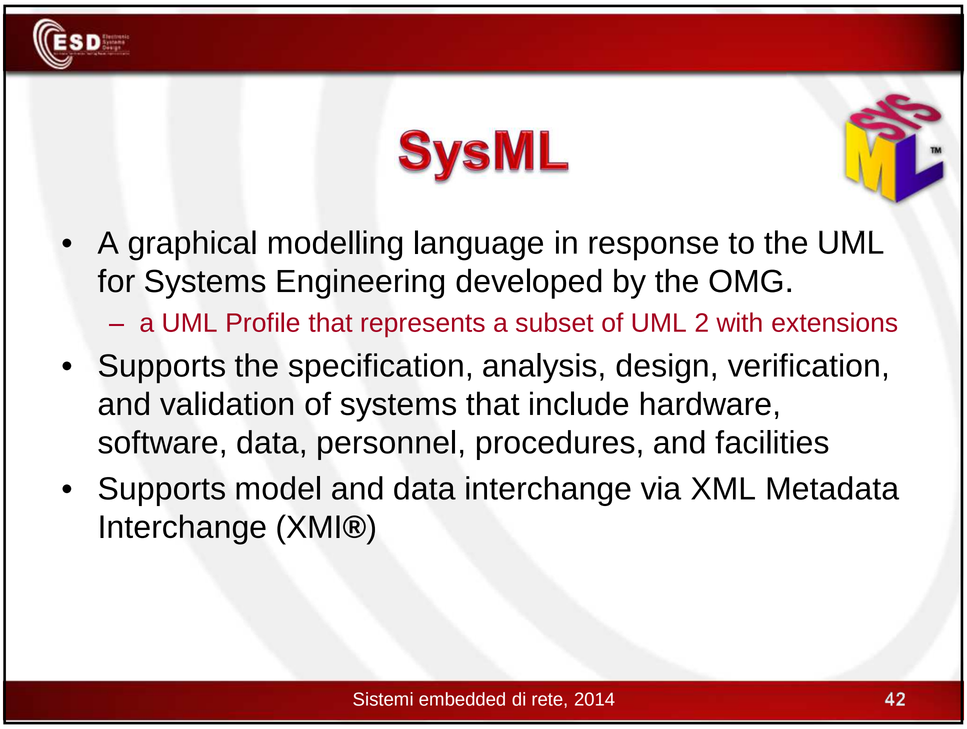
SysML



SysML

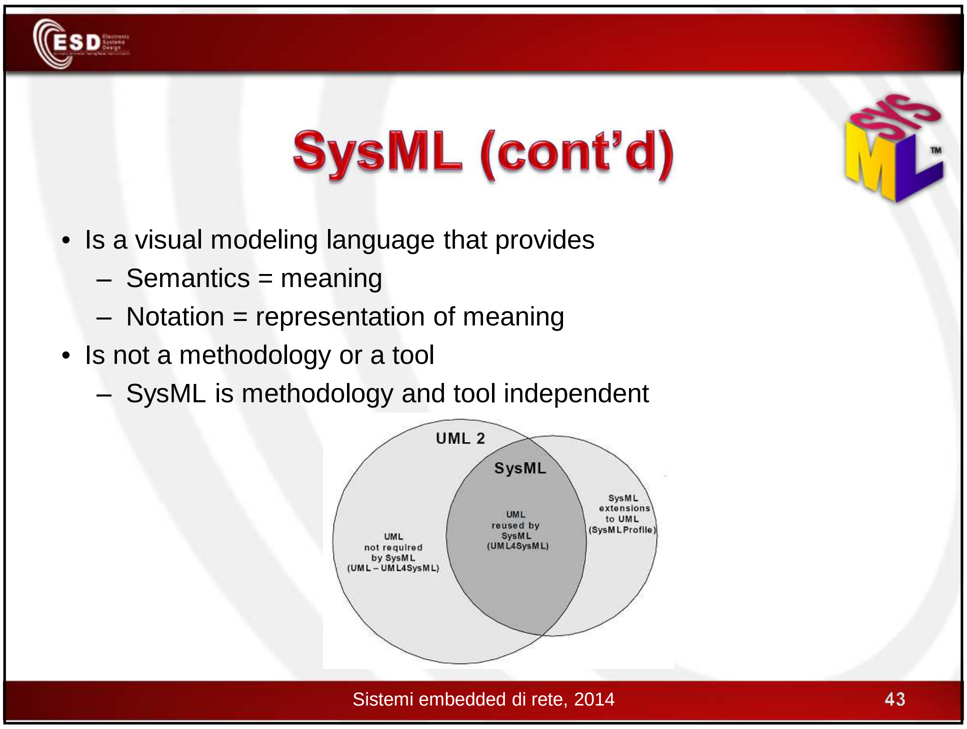
SysML

A Language to ***document*** the properties from different disciplines to ***describe*** the ***whole solution***



SysML

- A graphical modelling language in response to the UML for Systems Engineering developed by the OMG.
  - a UML Profile that represents a subset of UML 2 with extensions
- Supports the specification, analysis, design, verification, and validation of systems that include hardware, software, data, personnel, procedures, and facilities
- Supports model and data interchange via XML Metadata Interchange (XMI®)



SysML (cont'd)

- Is a visual modeling language that provides
  - Semantics = meaning
  - Notation = representation of meaning
- Is not a methodology or a tool
  - SysML is methodology and tool independent

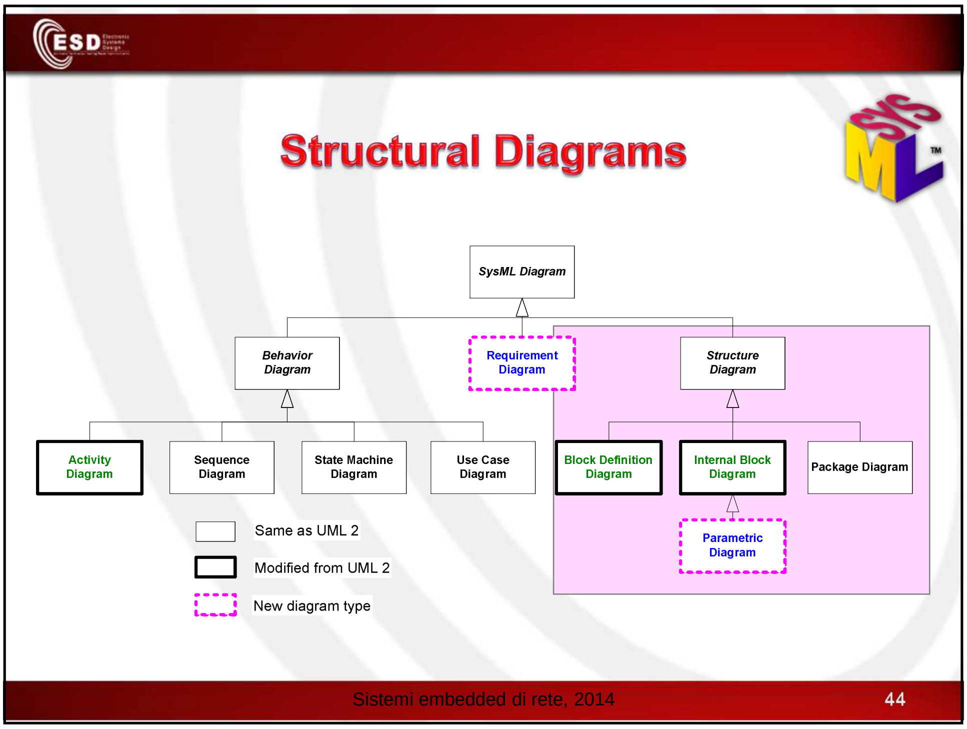
UML 2

SysML

UML not required by SysML (UML – UML4SysML)

UML reused by SysML (UML4SysML)

SysML extensions to UML (SysML Profile)



Structural Diagrams

SysML Diagram

Behavior Diagram

Requirement Diagram

Structure Diagram

Activity Diagram

Sequence Diagram

State Machine Diagram

Use Case Diagram

Block Definition Diagram

Internal Block Diagram

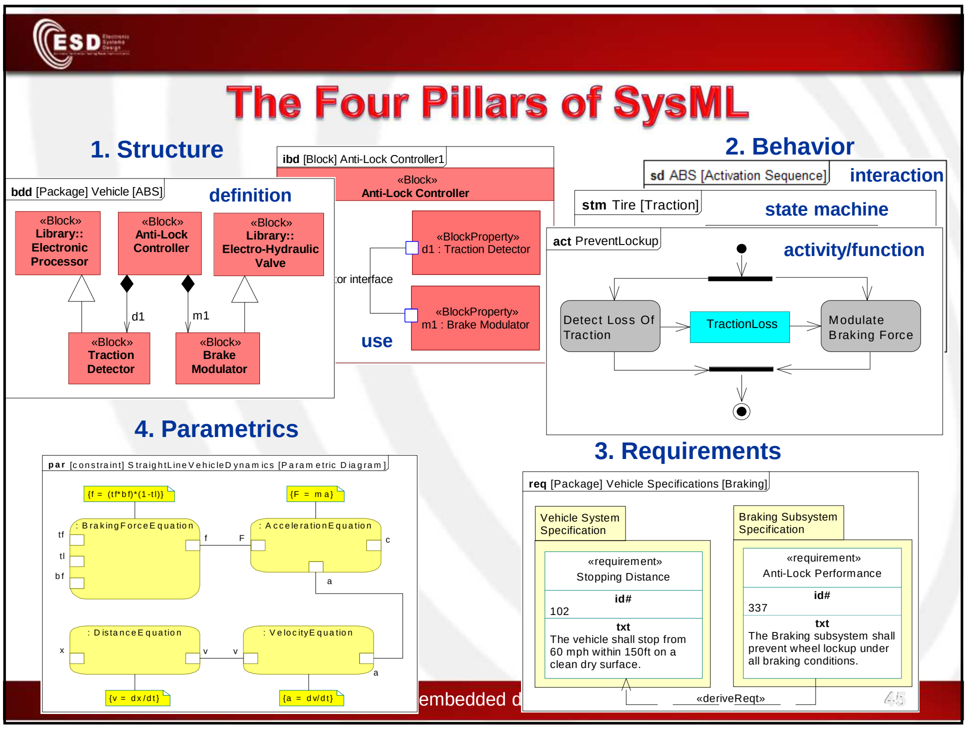
Package Diagram

Parametric Diagram

□ Same as UML 2

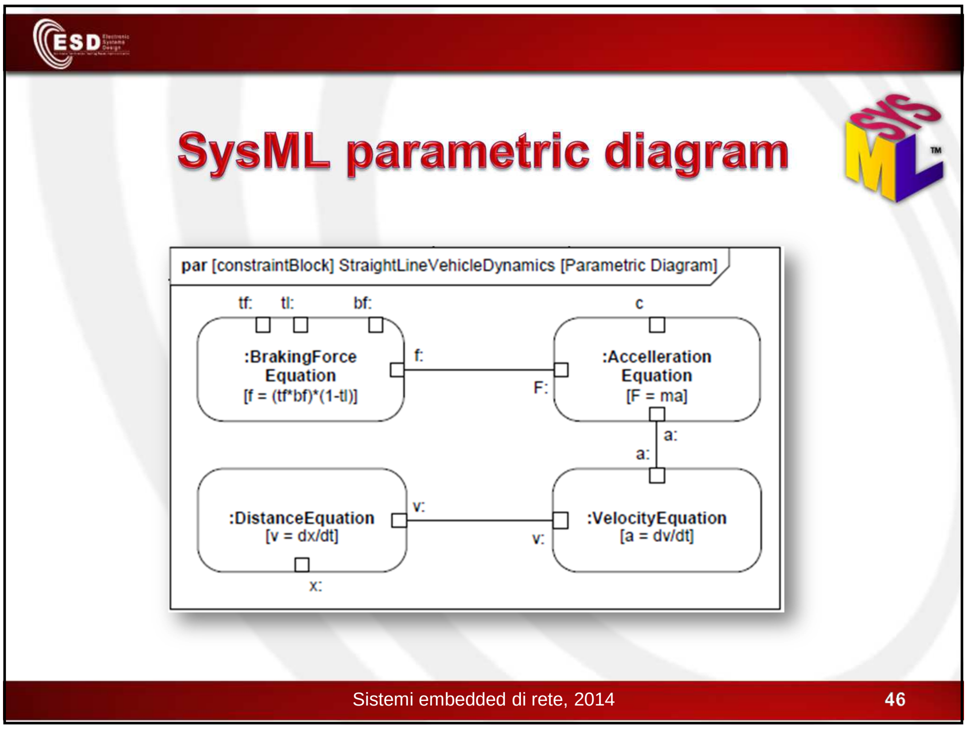
▭ Modified from UML 2

⬚ New diagram type



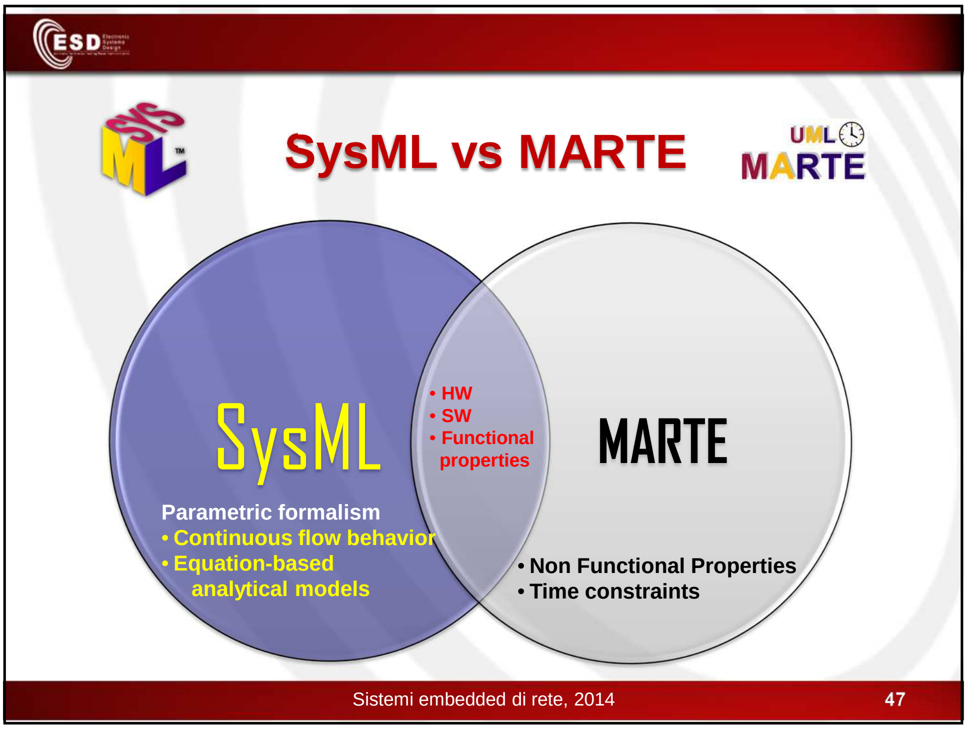
The Four Pillars of SysML

1. Structure
definition
bdd [Package] Vehicle [ABS]
«Block» Library:: Electronic Processor
«Block» Anti-Lock Controller
«Block» Library:: Electro-Hydraulic Valve
d1
m1
«Block» Traction Detector
«Block» Brake Modulator
ibd [Block] Anti-Lock Controller1
«Block» Anti-Lock Controller
«BlockProperty» d1 : Traction Detector
«BlockProperty» m1 : Brake Modulator
:or interface
use
2. Behavior
sd ABS [Activation Sequence]
interaction
stm Tire [Traction]
state machine
act PreventLockup
activity/function
Detect Loss Of Traction
TractionLoss
Modulate Braking Force
4. Parametrics
par [constraint] StraightLineVehicleDynamics [Parametric Diagram]
{f = (tf*bf)*(1-tl)}
{F = m a}
: BrakingForceEquation
: AccelerationEquation
tf
tl
bf
f
F
c
a
: DistanceEquation
: VelocityEquation
x
v
v
a
{v = dx/dt}
{a = dv/dt}
3. Requirements
req [Package] Vehicle Specifications [Braking]
Vehicle System Specification
Braking Subsystem Specification
«requirement» Stopping Distance
id# 102
txt The vehicle shall stop from 60 mph within 150ft on a clean dry surface.
«requirement» Anti-Lock Performance
id# 337
txt The Braking subsystem shall prevent wheel lockup under all braking conditions.
«deriveReqt»



SysML parametric diagram

par [constraintBlock] StraightLineVehicleDynamics [Parametric Diagram]
tf:
tl:
bf:
c
:BrakingForce Equation [f = (tf*bf)*(1-tl)]
f:
F:
:Accelleration Equation [F = ma]
a:
a:
:DistanceEquation [v = dx/dt]
v:
v:
:VelocityEquation [a = dv/dt]
x:



SysML vs MARTE

SysML

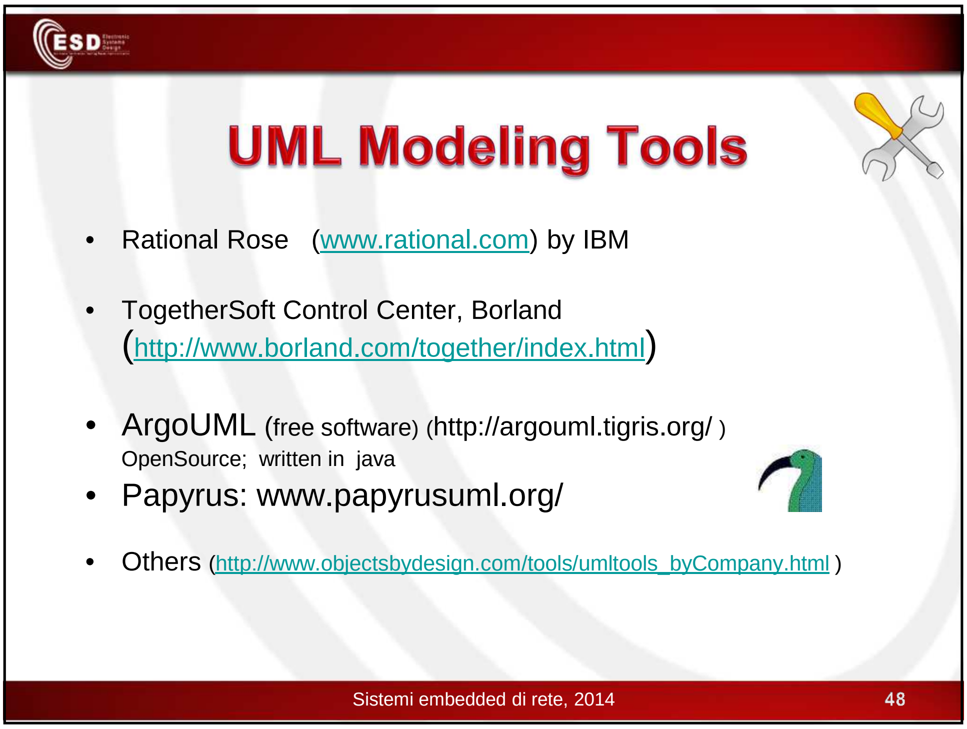
Parametric formalism
- Continuous flow behavior
- Equation-based analytical models

- HW
- SW
- Functional properties

MARTE

- Non Functional Properties
- Time constraints

Sistemi embedded di rete, 2014 47



UML Modeling Tools

- Rational Rose (www.rational.com) by IBM
- TogetherSoft Control Center, Borland (http://www.borland.com/together/index.html)
- ArgoUML (free software) (http://argouml.tigris.org/) OpenSource; written in java
- Papyrus: www.papyrusuml.org/
- Others (http://www.objectsbydesign.com/tools/umltools_byCompany.html)

Sistemi embedded di rete, 2014 48

Reference

1. **UML Distilled:** A Brief Guide to the Standard Object Modeling Language
   Martin Fowler, Kendall Scott

2. IBM Rational
   http://www-306.ibm.com/software/rational/uml/

3. Practical UML --- A Hands-On Introduction for Developers
   http://www.togethersoft.com/services/practical_guides/umlonlinecourse/

4. Software Engineering Principles and Practice. Second Edition;
   Hans van Vliet.

5. http://www-inst.eecs.berkeley.edu/~cs169/