

LEZIONI 45-46

METAMODELLAZIONE, METACLASSI, E REFLECTION

... UML e Java

Ingegneria del Software e Progettazione Web
Università degli Studi di Tor Vergata - Roma

Guglielmo De Angelis
guglielmo.deangelis@isti.cnr.it

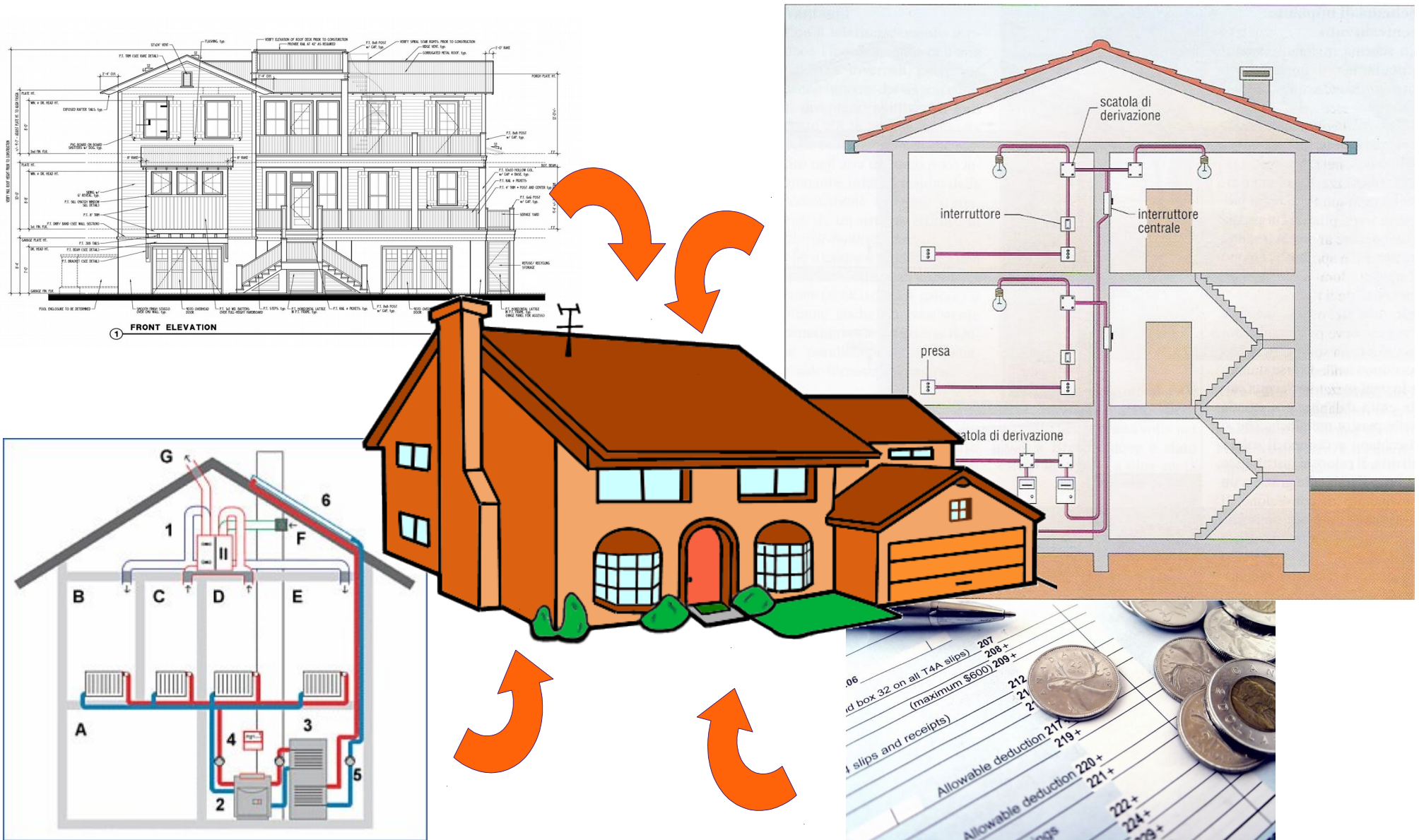
linguaggi di modellazione ... “at a glance”

- i linguaggi di modellazione sono spesso framework concettuali
- il loro obiettivo è supportare i progettisti nella
 - formalizzazione dei loro pensieri
 - rappresentare la realtà (o una sua particolare visione)

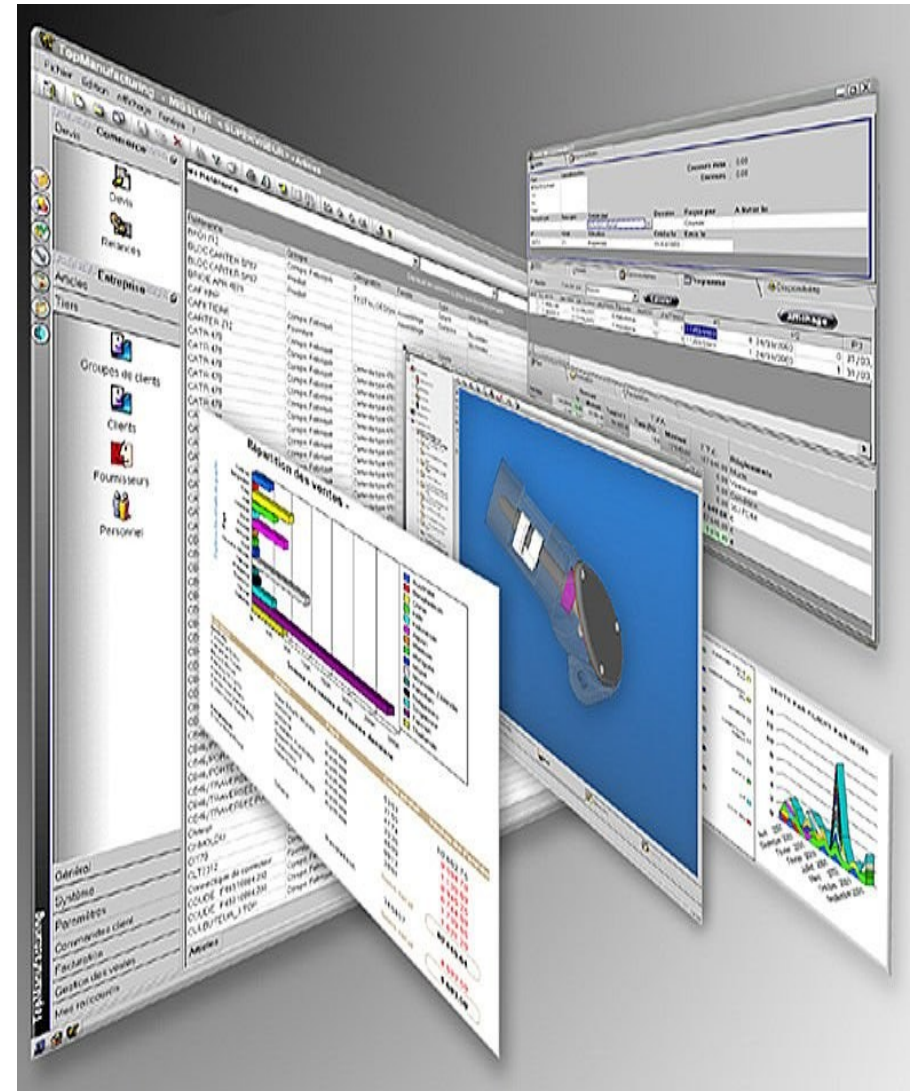
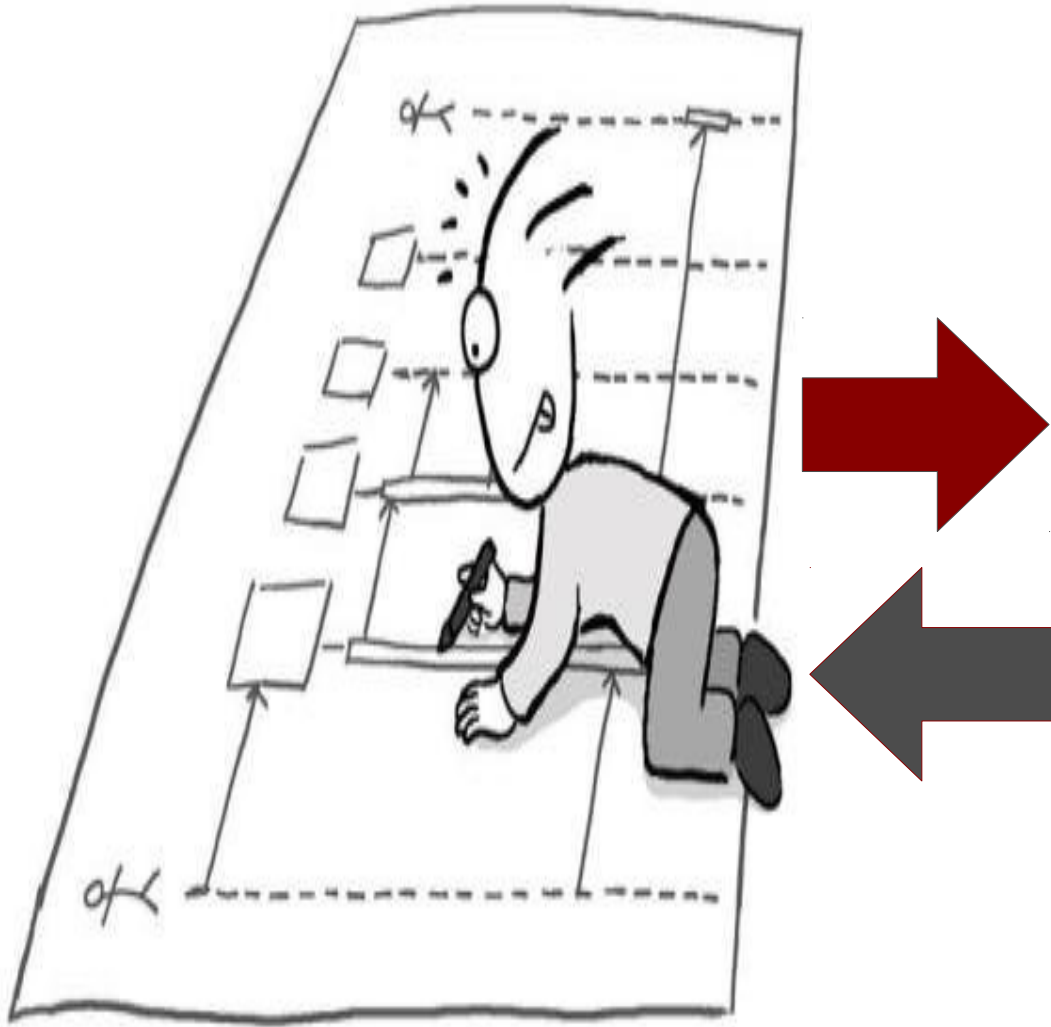
linguaggi di modellazione ... “at a glance”

- in generale consentono di modellare vari aspetti di un sistema
 - e.g. offrendo una combinazione di viste linkabili
- tipicamente ogni aspetto può essere descritto con uno o più specifiche/diagrammi che possono
 - usare vari simboli/notazioni
 - formale o semi-formale
 - testuale, grafica, o entrambe
 - focalizzarsi su differenti rappresentazioni del problema e della soluzione
- comunemente questi aspetti sono classificati in
 - statici (o strutturali) : elementi e loro relazioni
 - dinamici : azioni o interazioni relative agli elementi modellati

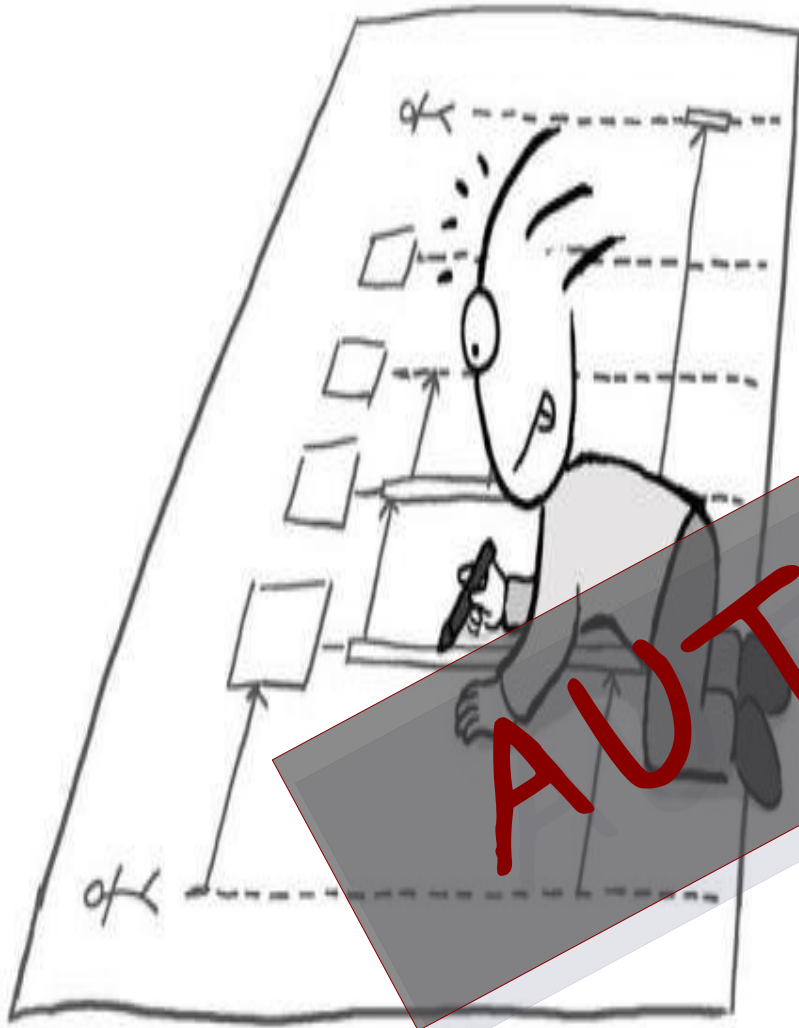
torniamo all'esempio della casa ...



IdS && modelli : rappresentare,
analizzare, sintetizzare



IdS && modelli : automazione



AUTOMATIC



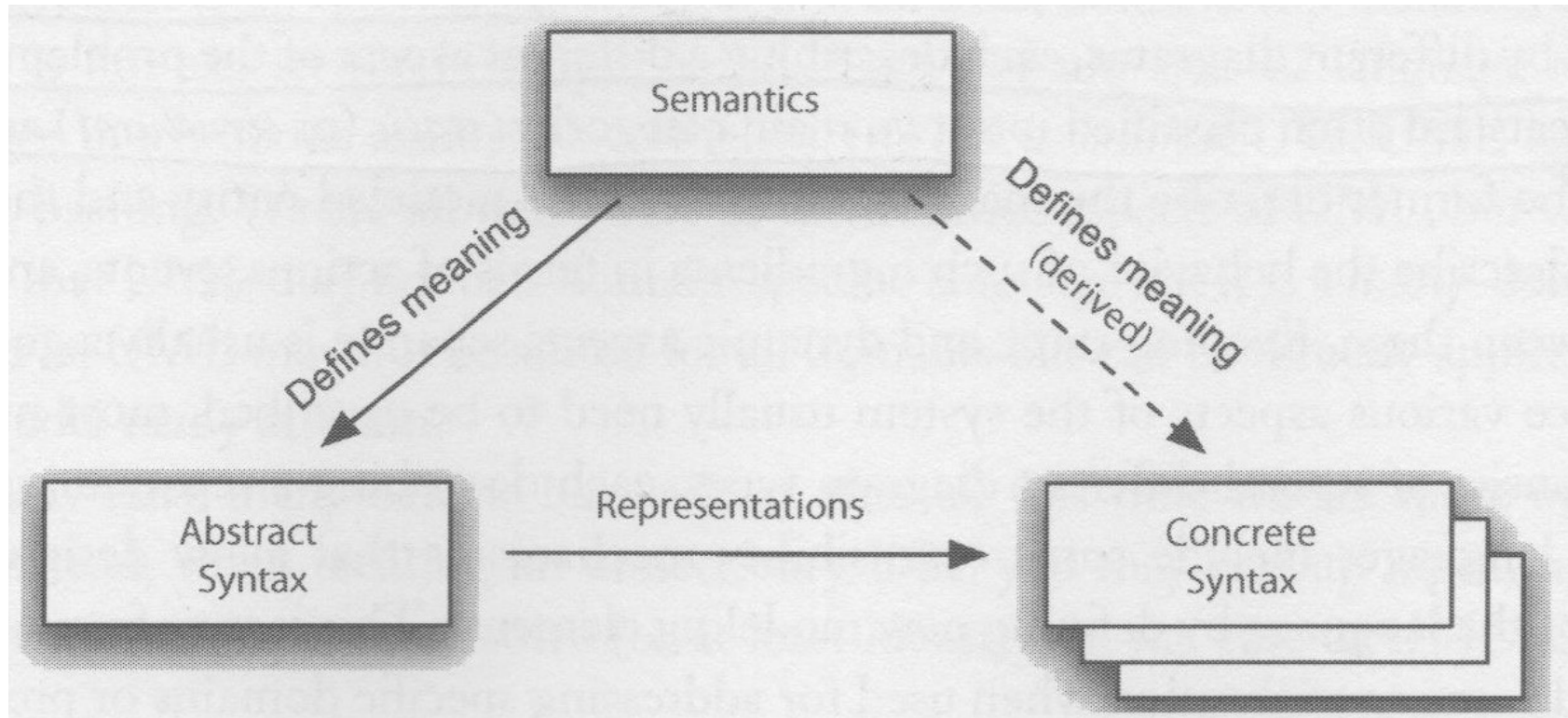
- come manipolare in modo automatico modelli software?

—

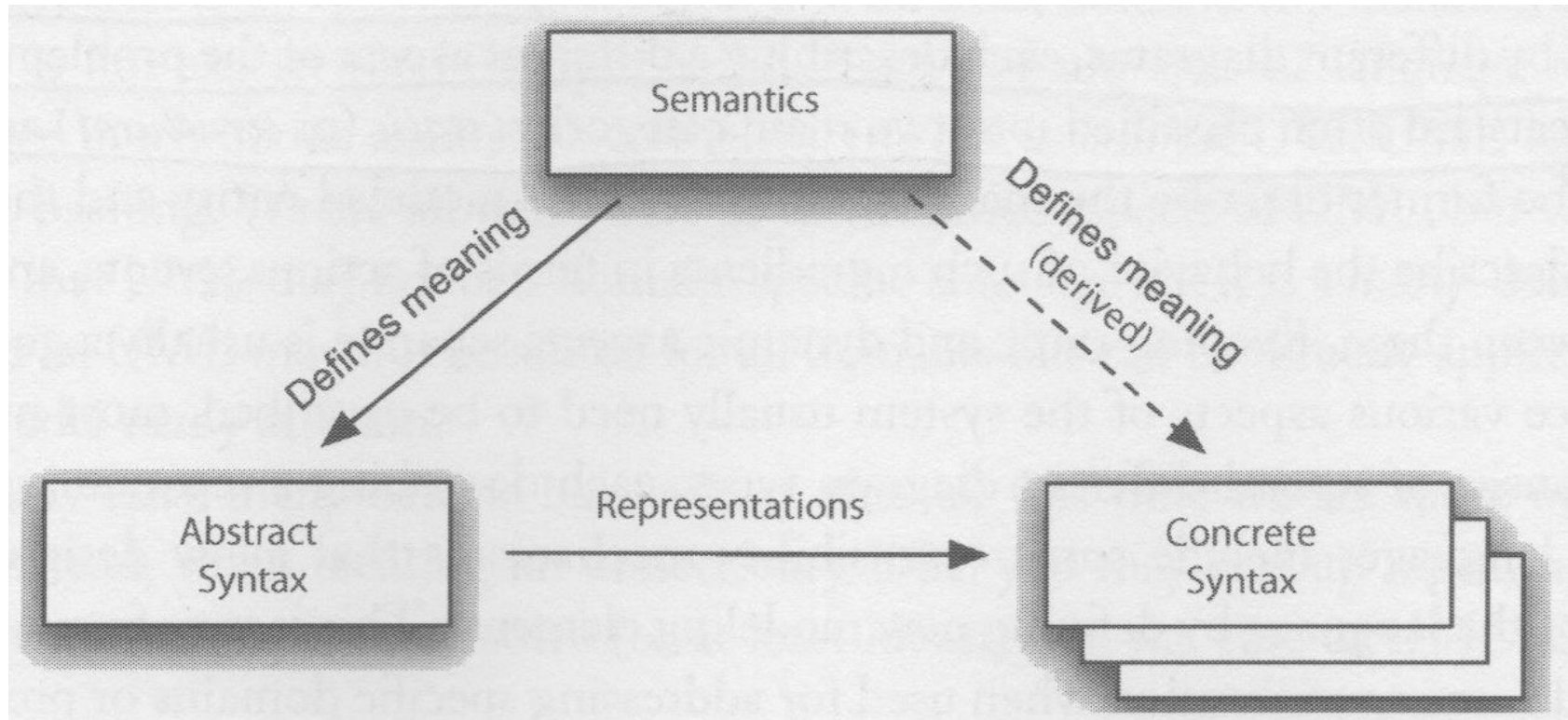
—

anatomia dei linguaggi di modellazione

anatomia dei linguaggi di modellazione

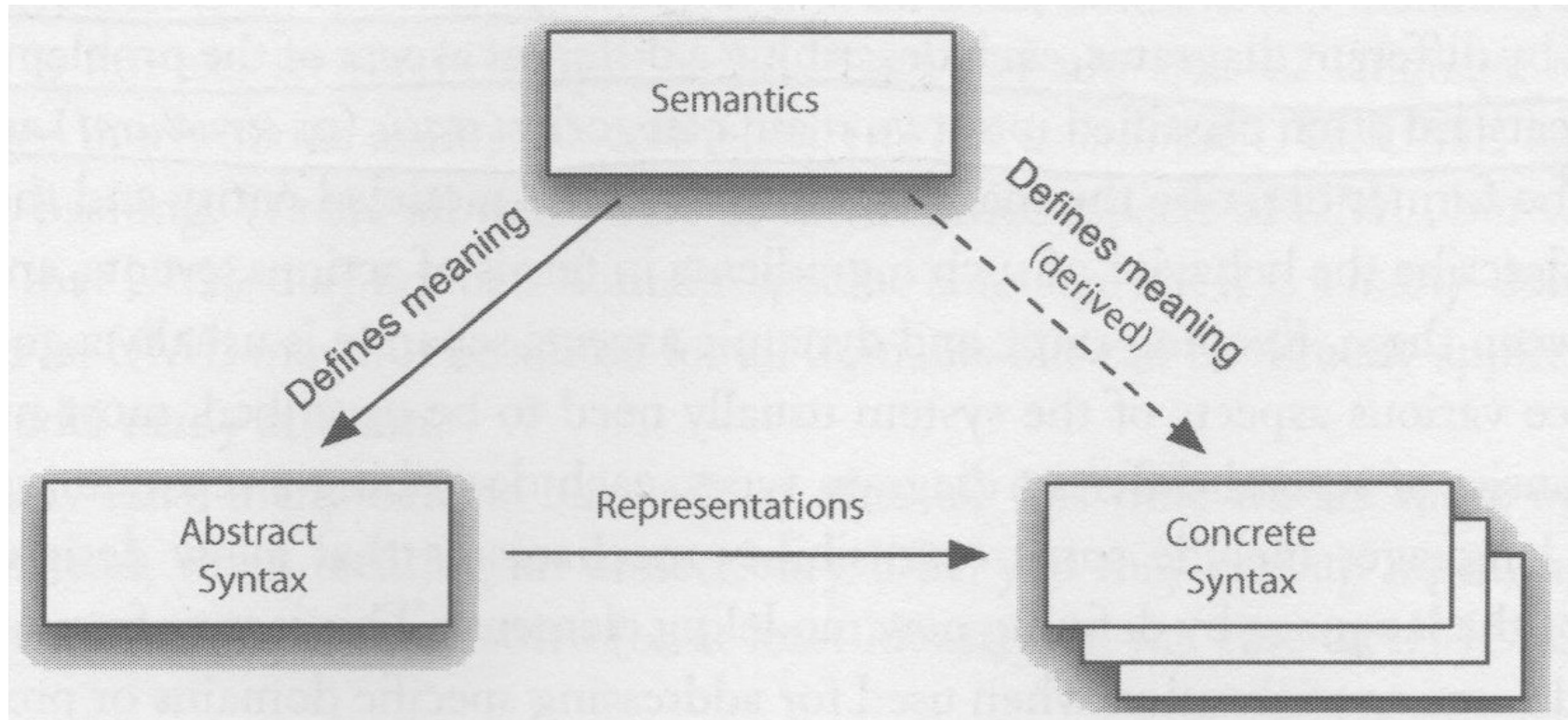


anatomia dei linguaggi di modellazione



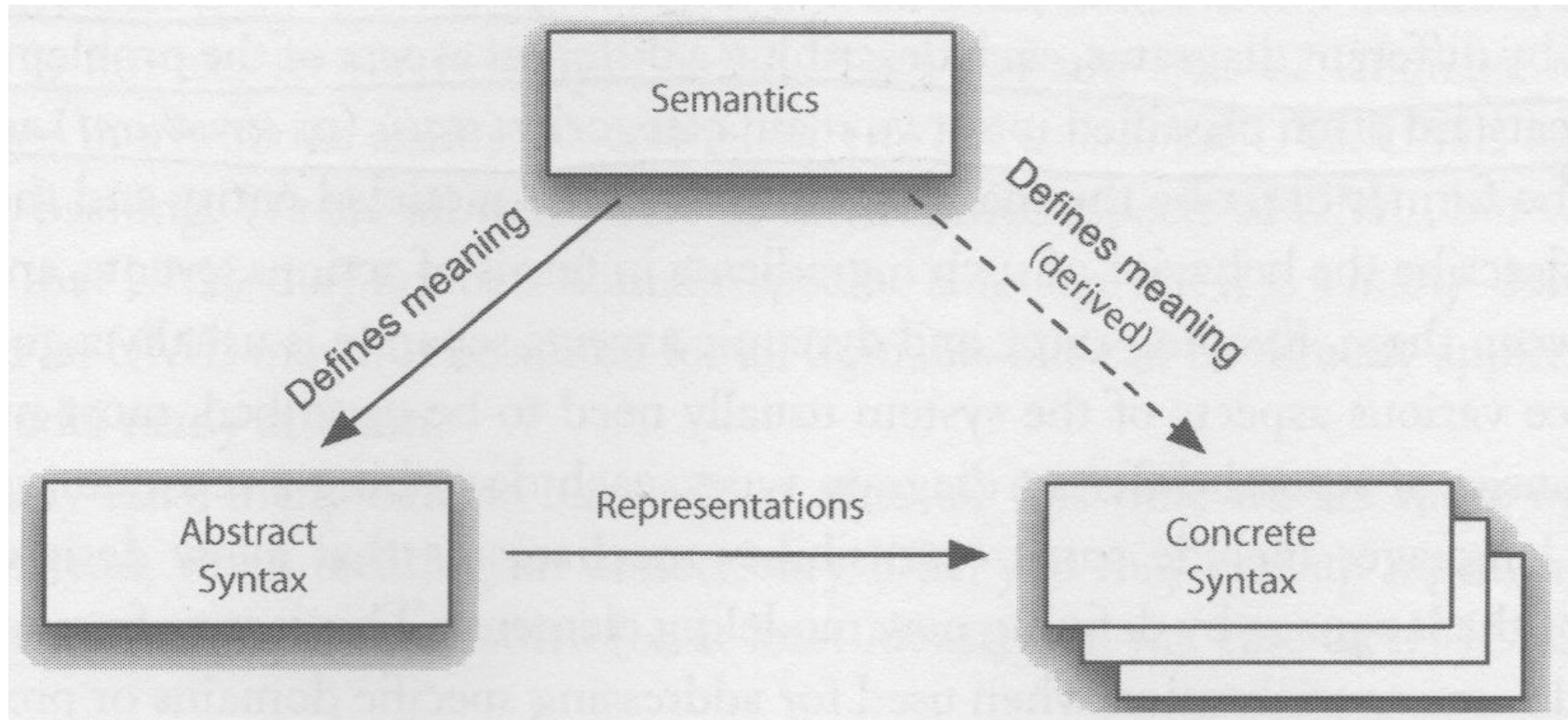
- **abstract syntax**: describe la struttura del linguaggio, ed il modo in cui le primitive possono essere combinate tra loro. E' indipendente da ogni tipo di rappresentazione/codifica

anatomia dei linguaggi di modellazione



- **concrete syntax**: describe la specifica rappresentazione di un linguaggio; la notazione può essere sia testuale che grafica. E' utilizzata dai progettisti per creare modelli.

anatomia dei linguaggi di modellazione



- **semantics**: definisce il significato di ogni elemento del linguaggio. Può essere una definizione sia formale che semi-formale. Una parziale o scorretta specifica della semantica causa incomprensioni ed usi scorretti del linguaggio.

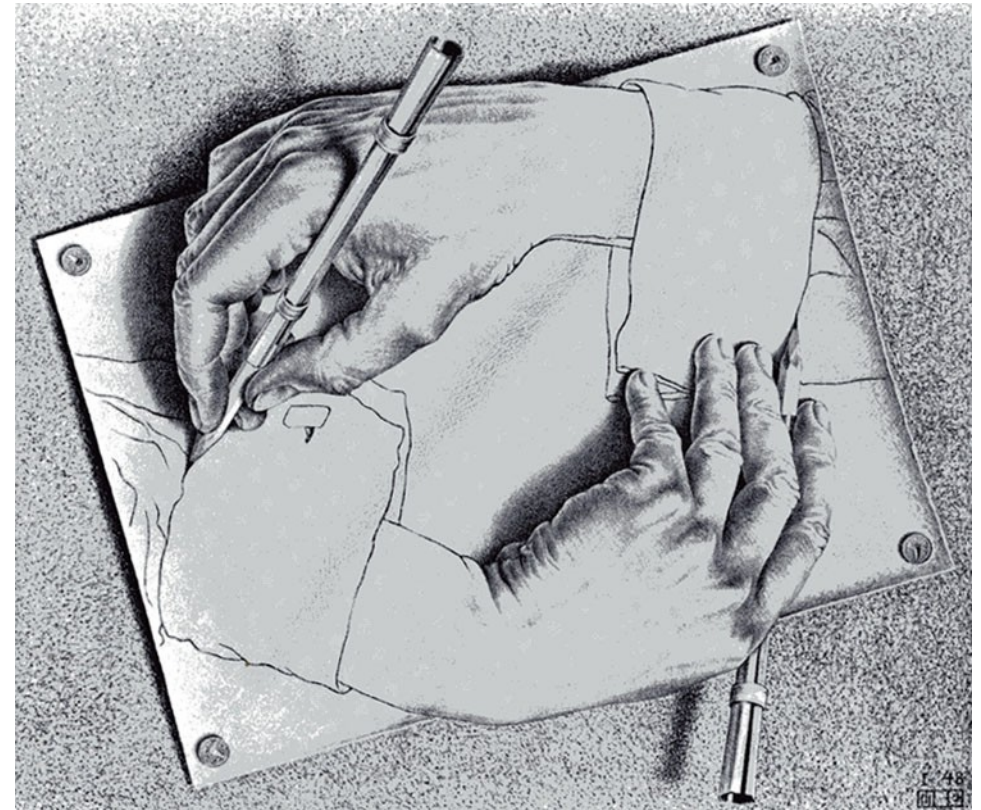
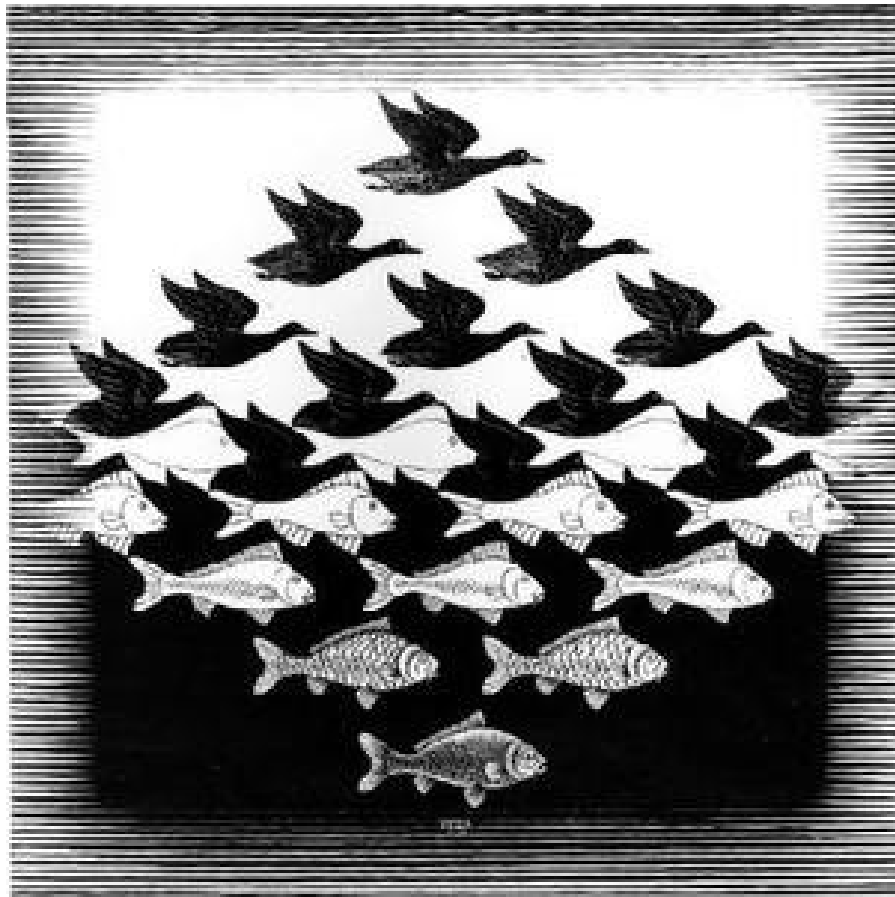
- come manipolare in modo automatico modelli software?

—

—

- come manipolare in modo automatico modelli software?

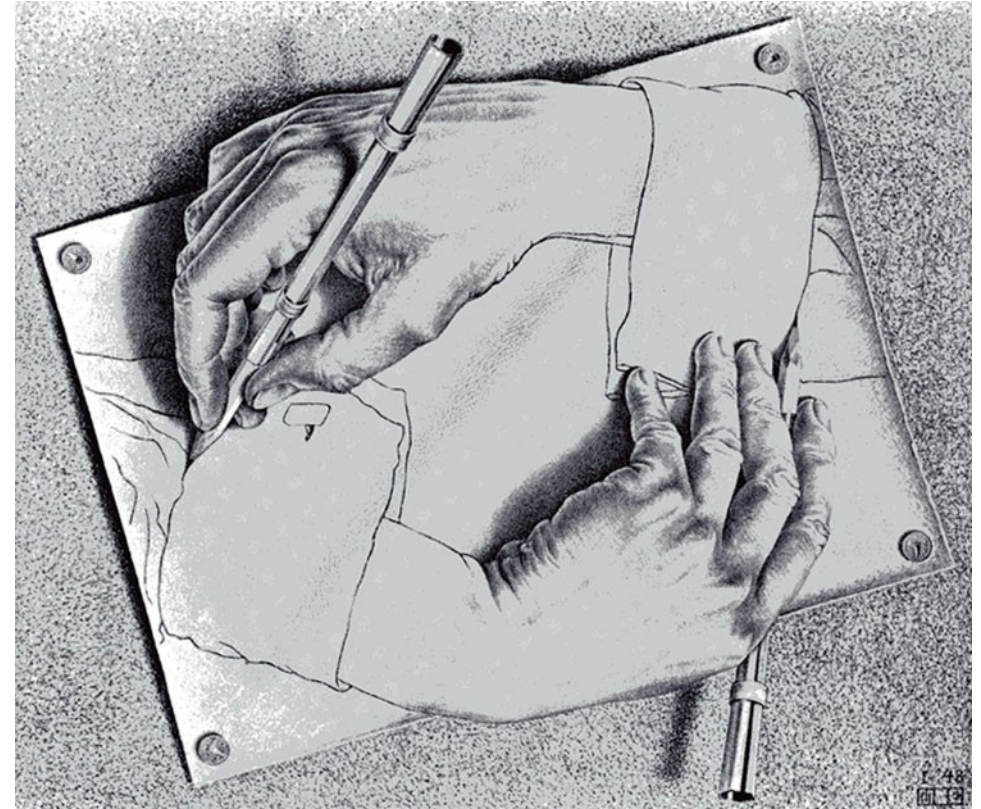
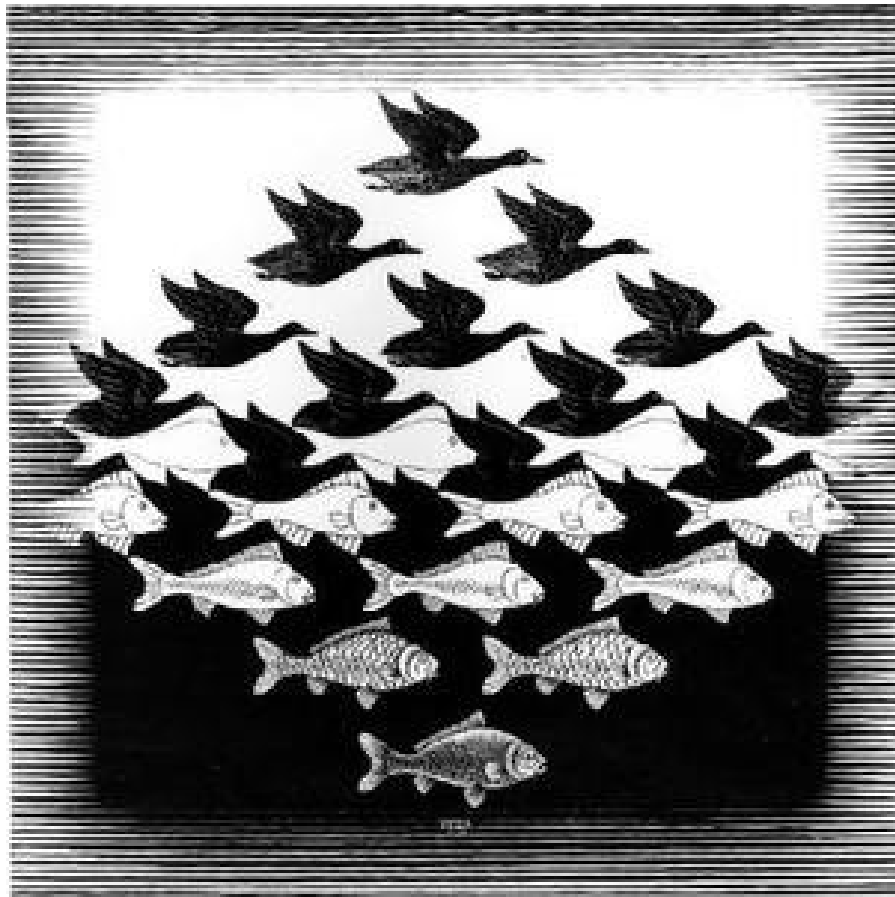
- meta-modellazione



- trasformazioni (automatiche)

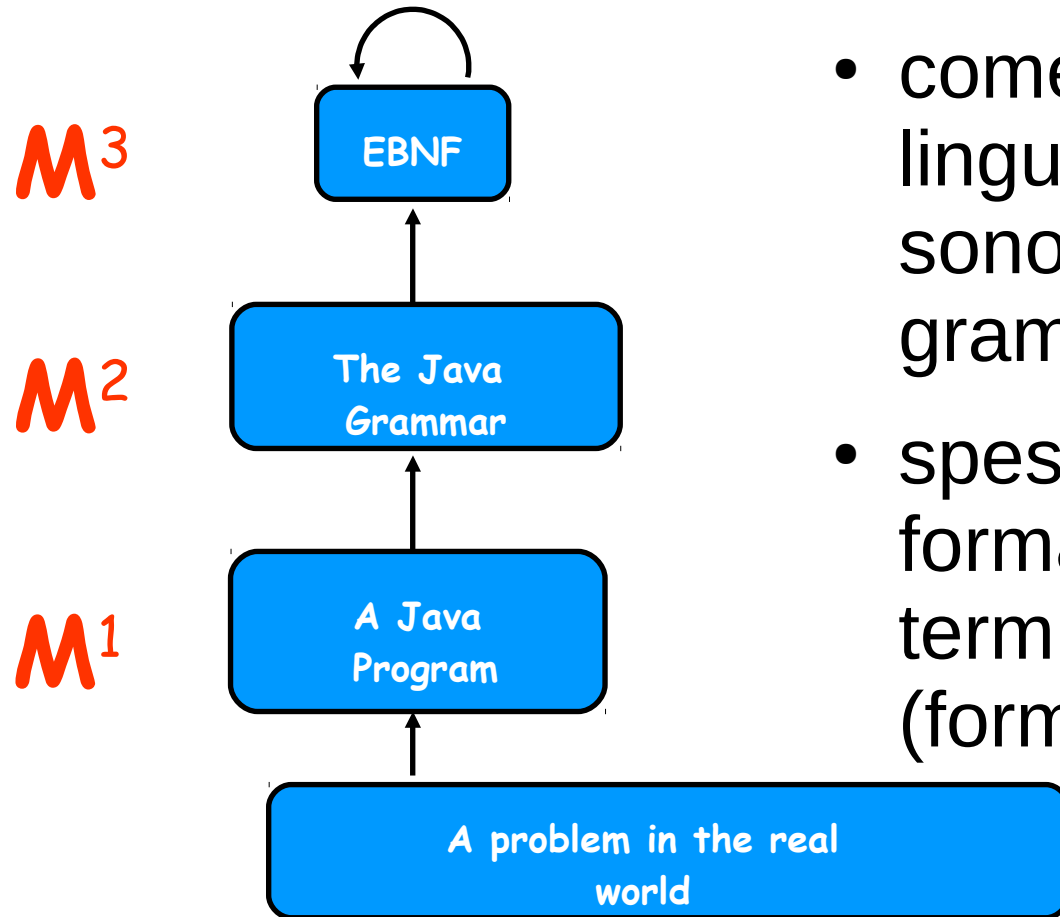
- come manipolare in modo automatico modelli software?

– meta-modellazione



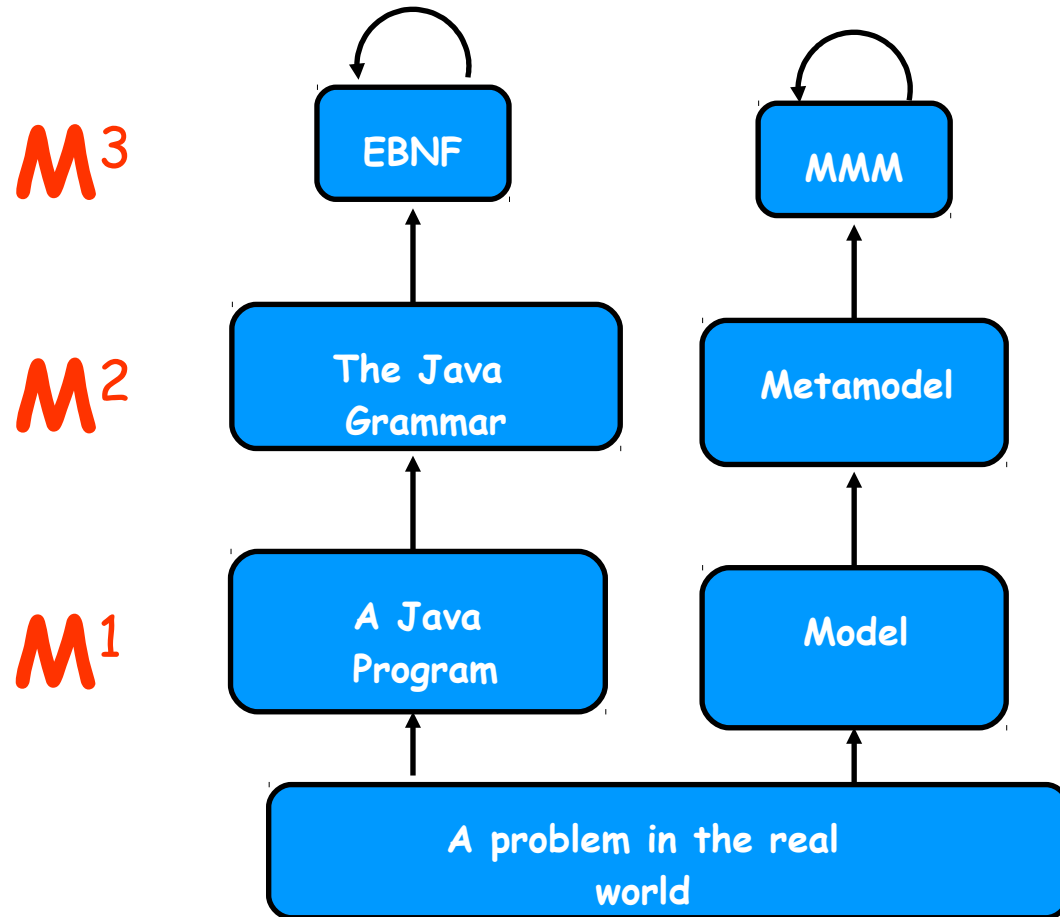
– trasformazioni (automatiche)

dalla teoria dei linguaggi ... - 1 -



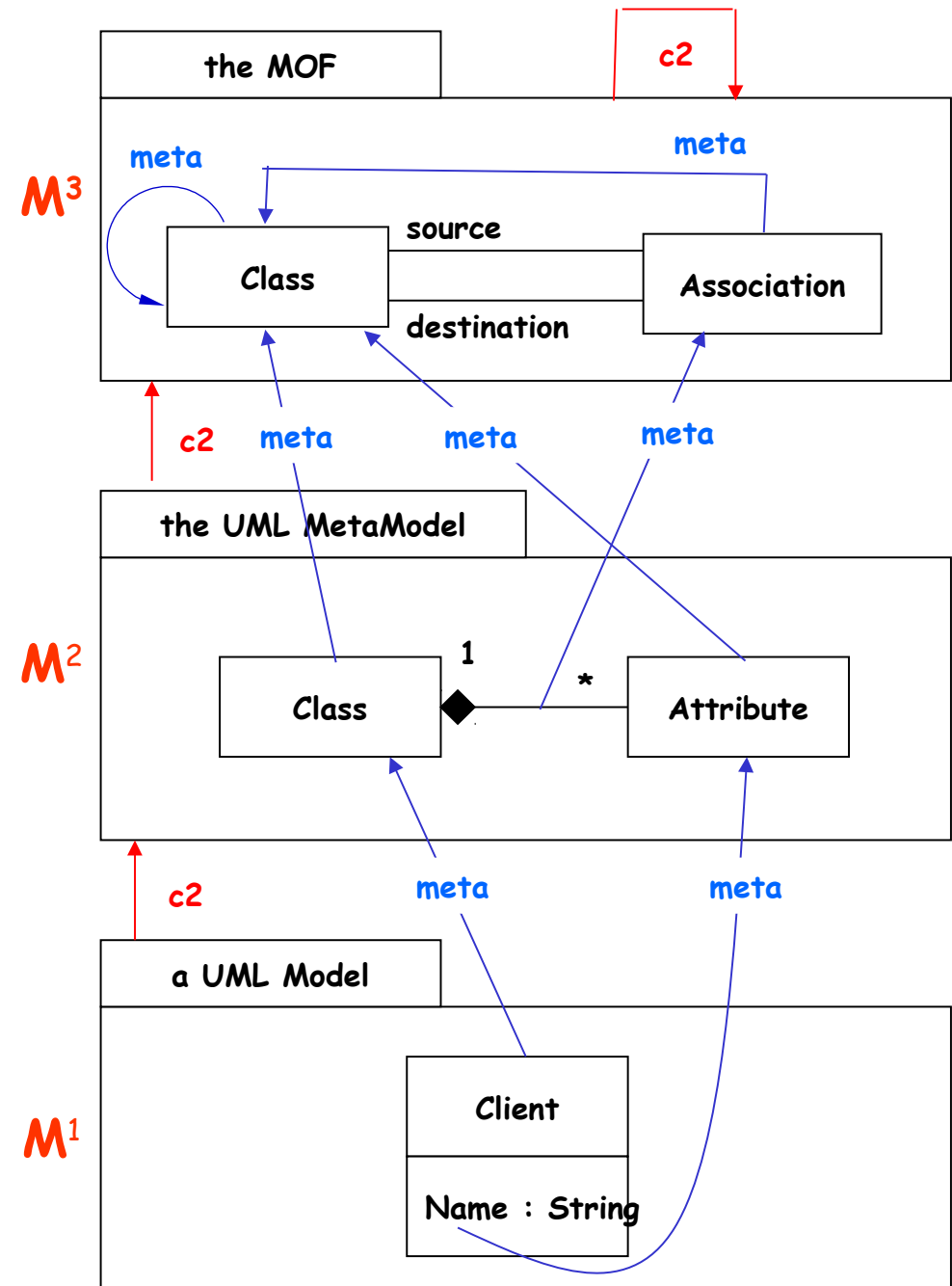
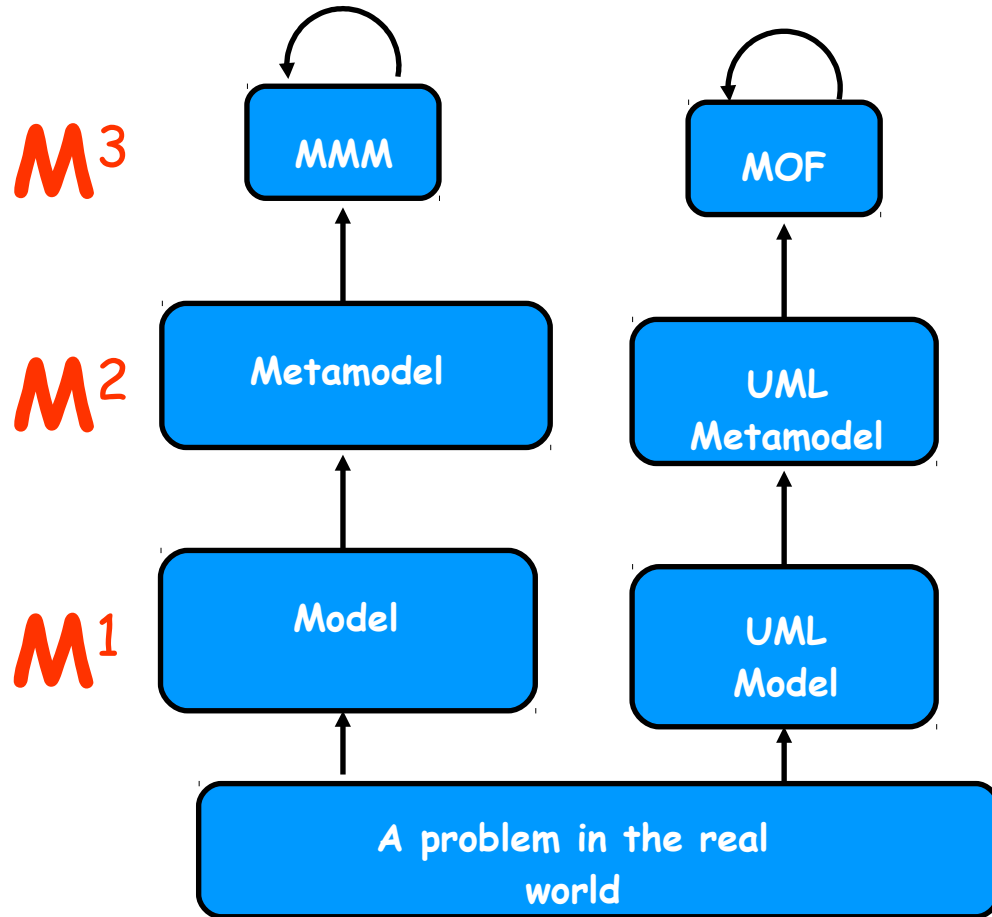
- come in linguaggi naturali, i linguaggi di programmazione sono definiti attraverso grammatiche formali
- spesso, queste grammatiche formali sono espresse in termini di altri linguaggi (formali)

... la stessa cosa avviene per i linguaggi di modellazione



- i modelli sono conformi alla “grammatica” del linguaggio (i.e. metamodello)
- il metamodello dichiara (almeno) la sintassi astratta del linguaggio di modellazione
- i metamodelli sono definiti per mezzo di altri linguaggi chiamati : meta-metamodelli

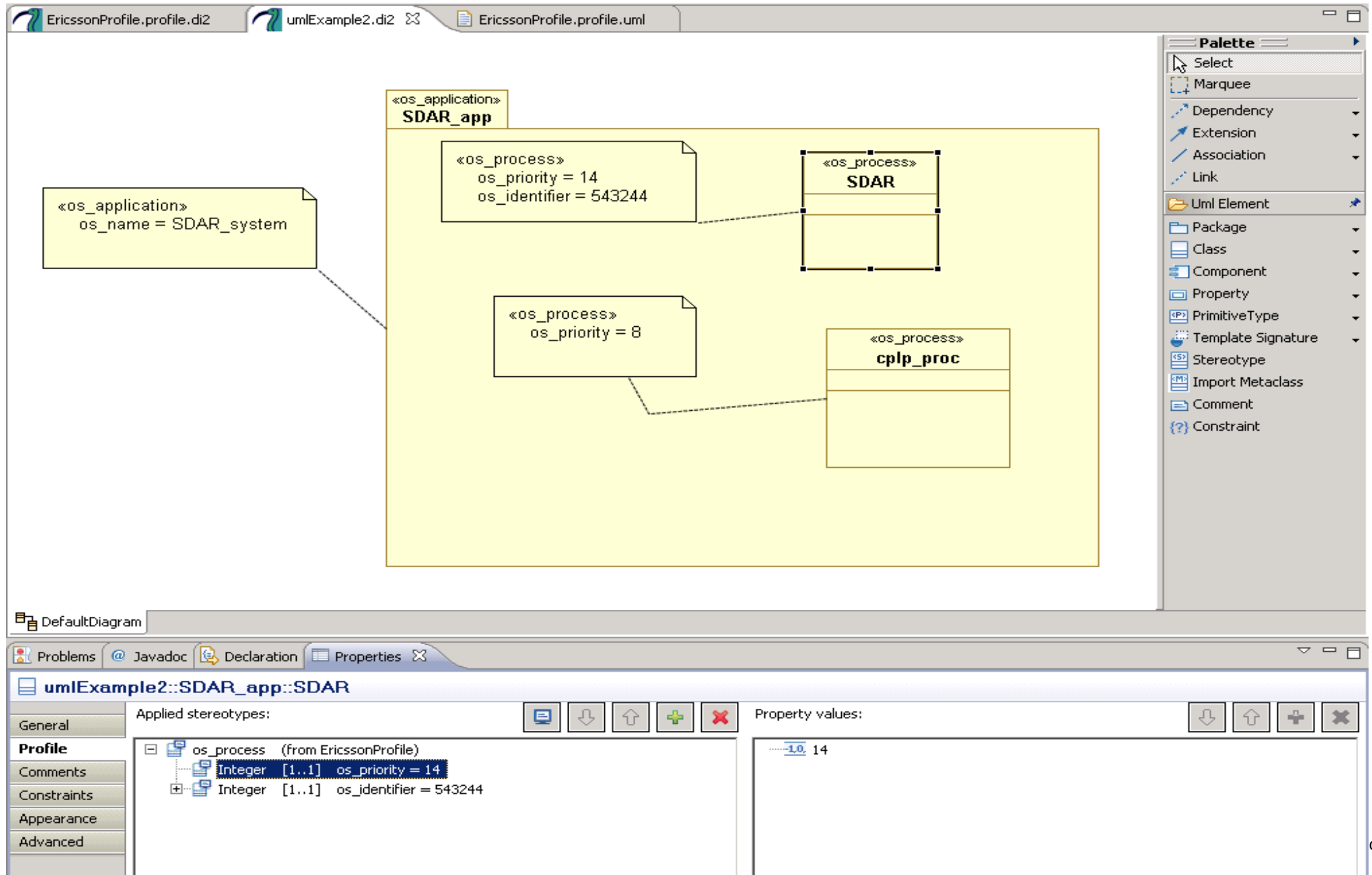
metamodeling



disegnare **VS** modellare

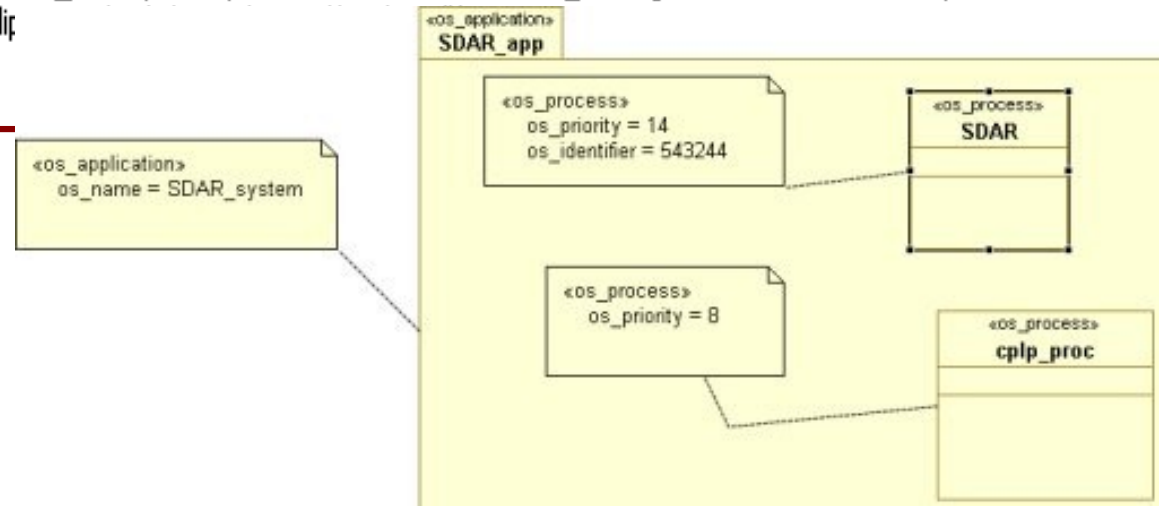
- strumenti per disegnare
 - non riferiscono ad alcuna grammatica per i modelli trattati
 - non è associato nessun significato agli elementi modellati
 - gestiscono la rappresentazione grafica degli elementi del modello
 - forme, linee, frecce
 - non c'è controllo sulle relazioni tra gli elementi modellati
- strumenti per modellare
 - rappresentano elementi secondo una grammatica
 - definiscono relazioni tra elementi solo se previste dalla grammatica

drawing VS modeling – example : graphic notation



drawing VS modeling – example : textual notation

```
<contents xmi:type="ecore:EPackage" xmi:id="_jxPwEZ3MEdyEi-YIP6R6ew" name="EricssonProfile" nsURI="http://schemas/EricssonProfile/" xmi:isRoot="true">
  <eClassifiers xmi:type="ecore:EClass" xmi:id="_jxPwEp3MEdyEi-YIP6R6ew" name="os_process">
    <eAnnotations xmi:id="_jxPwE53MEdyEi-YIP6R6ew" source="http://www.eclipse.org/uml2/2.0.0/UML" references="_jIN3QJ3LEdyEi-YIP6R6ew"/>
    <eStructuralFeatures xmi:type="ecore:EAttribute" xmi:id="_jxPwFJ3MEdyEi-YIP6R6ew" name="os_priority" ordered="false" unique="false" lowerBound="1">
      <eType xmi:type="ecore:EDatatype" href="http://www.eclipse.org/emf/2002/Ecore#//EInt"/>
    </eStructuralFeatures>
    <eStructuralFeatures xmi:type="ecore:EReference" xmi:id="_jxPwFp3MEdyEi-YIP6R6ew" name="base_Class" ordered="false" unique="false" lowerBound="1">
      <eType xmi:type="ecore:EClass" href="http://www.eclipse.org/uml2/2.1.0/UML#//Class"/>
    </eStructuralFeatures>
    <eStructuralFeatures xmi:type="ecore:EAttribute" xmi:id="_jxPwGJ3MEdyEi-YIP6R6ew" name="os_identifier" ordered="false" unique="false" lowerBound="1">
      <eType xmi:type="ecore:EDatatype" href="http://www.eclipse.org/emf/2002/Ecore#//EInt"/>
    </eStructuralFeatures>
  </eClassifiers>
  <eClassifiers xmi:type="ecore:EClass" xmi:id="_jxPwGp3MEdyEi-YIP6R6ew" name="os_application">
    <eAnnotations xmi:id="_jxPwG53MEdyEi-YIP6R6ew" source="http://www.eclipse.org/uml2/2.0.0/UML" references="_j_cVIJ3LEdyEi-YIP6R6ew"/>
    <eStructuralFeatures xmi:type="ecore:EAttribute" xmi:id="_jxPwHJ3MEdyEi-YIP6R6ew" name="os_name" ordered="false" unique="false" lowerBound="1">
      <eType xmi:type="ecore:EDatatype" href="http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    </eStructuralFeatures>
    <eStructuralFeatures xmi:type="ecore:EReference" xmi:id="_jxPwHp3MEdyEi-YIP6R6ew" name="base_Package" ordered="false" unique="false" lowerBound="1">
      <eType xmi:type="ecore:EClass" href="http://www.eclipse.org/uml2/2.1.0/UML#//Package"/>
    </eStructuralFeatures>
  </eClassifiers>
</contents>
```



- è possibile sfruttare queste nozioni nei linguaggi di modellazione/programmazione
- come?

reflection

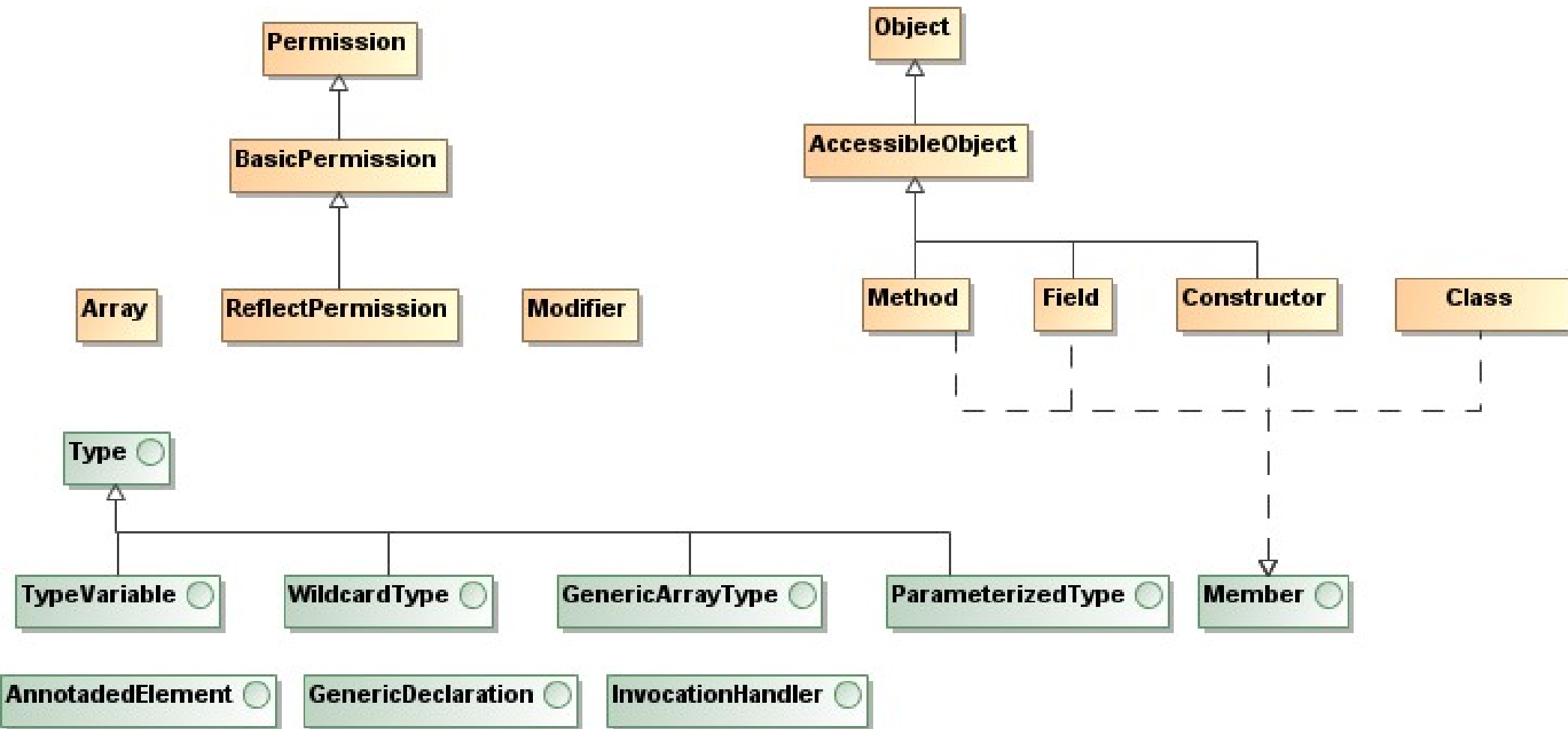
- meccanismo abilitato a tempo di esecuzione che consente :
 - analizzare la struttura di una classe
 - richiedere la creazione dinamica di oggetti
 - richiedere dinamicamente l'invocazione di metodi

come si sfrutta in Java?

la classe Object

- IN JAVA TUTTO (o quasi) è un OGGETTO!!
- Object è una classe
 - è il tipo base di ogni classe definita in Java
- se una classe non estende esplicitamente da un'altra classe, allora implicitamente estende Object
 - non c'è bisogno di “extends Object”
 - quindi ogni classe “is-a-kind-of” Object
- Object offre metodi basilari che sono ereditati da tutte le classi
 - `boolean equals(Object other); /* checks whether two object are the same from a semantics standpoint (to check if they are exactly the same object in memory, use == */`
 - `Object clone(); // instantiates a new object with the same values`
 - `void finalize(); // protected, cleans up memory and resources`
 - `int hashCode(); // unique (almost) identifier of the object`
 - `String toString(); // returns the class name + object hash code`

reflection model in Java



la *classe* Class – 1

- “istanze” della classe Class rappresentano classi ed interfacce in esecuzione in una applicazione Java
 - una enumerazione Java (i.e. enum) è rappresentato da un oggetto di tipo Class
 - una annotazione Java (i.e. @) è rappresentato da un oggetto di tipo Class
 - un array è rappresentato da un oggetto di tipo Class (una istanza per coppia [tipo-array,dimensione])
 - i tipi primitivi di Java (boolean, byte, char, short, int, long, float, and double), e la keyword “void” are rappresentato da oggetti di tipo Class

la *classe* Class – 2

- Class non definisce un costruttore pubblico:
 - i.e. non si può fare la “new” su un “Class”
- gli oggetti di tipo Class sono istanziati automaticamente dalla JVM e caricati in memoria
 - su questo aspetto ci sono alcune controversie che contribuiscono a non far considerare Java un linguaggio puramente orientato agli oggetti

esempi reflection – 1

VEDERE MATERIALE ALLEGATO
ALLA LEZIONE :

`ReflectionExample.java`

dynamic loading

- alla esecuzione di un programma Java, la prima classe che viene caricata è quella il cui metodo “main” è stato invocato
 - solo successivamente vengono caricate in memoria le altre classi riferite
- cosa accade sotto il cofano se si cerca di istanziare una classe che non presente in memoria?

dynamic loading

- alla esecuzione di un programma Java, la prima classe che viene caricata è quella il cui metodo “main” è stato invocato
 - solo successivamente vengono caricate in memoria le altre classi riferite
- cosa accade sotto il cofano se si cerca di istanziare una classe che non presente in memoria?
 - entriamo in logica di errore
 - banalmente: viene sollevata l'eccezione “ClassNotFoundException”
 - soluzione: viene catturato e gestito e l'errore chiedendo dinamicamente al ClassLoader di caricare la classe
 - come?
 - attraverso “Class.forName(classFullNameString)”

dynamic loading

- alla esecuzione di un programma Java, la prima classe che viene caricata è quella il cui metodo “main” è stato invocato
 - solo successivamente vengono caricate in memoria le altre classi riferite
- cosa accade sotto il cofano se si cerca di istanziare una classe che non presente in memoria?
 - entriamo in logica di errore
 - banalmente: viene sollevata l'eccezione “ClassNotFoundException”
 - soluzione: viene catturato e gestito e l'errore chiedendo dinamicamente al ClassLoader di caricare la classe
 - come?
 - attraverso “Class.forName(classFullNameString)”
- Dynamic Java Class è una importante funzionalità offerta dalla JVM perchè consente di installare componenti software a run-time e on-the-fly
 - principio del “lazy loading”: rinvio del caricamento delle classi in memoria all'ultimo momento possibile, solo quando servono effettivamente

dynamic loading – perchè nella pratica?

The screenshot shows a web browser window with the title "Why do we use only dynamic class loading for JDBC drivers | Oracle Community - Mozilla Firefox". The address bar shows the URL "https://community.oracle.com/thread/2084971". The page header includes the Oracle logo, "Oracle Community Directory", "Oracle Community FAQ", and a "Log in" link. A search bar is also present.

The main content area shows a thread titled "2. Re: Why do we use only dynamic class loading for JDBC drivers" by user "796447" on "Oct 31, 2007 12:48 PM (In response to 807603)". The user's profile picture and "Level 1" badge are visible.

The post content is as follows:

chandunitw wrote:
Hi,

My JDBC experience is that we always use dynamic class loader for drivers.
If we have a well defined package from a vendor, why do we use this dynamic class loading feature for drivers??

Often times, the driver class name is set in a configuration file, not in code. So the thing which processes the configuration file has no idea ahead of time which driver or drivers it will support, so it is not coded specifically for any. So it loads the driver by reflection, since it is given the class name as a string it can use with the reflection API.

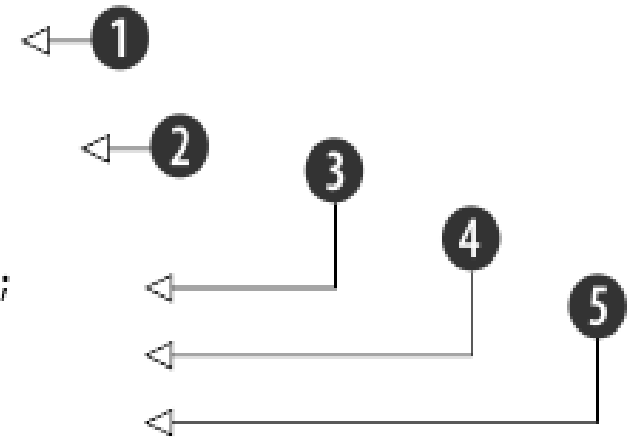
At the bottom right of the post, there is a link "Go to original post".

il mio primo programma JUnit

```
import static org.junit.Assert.*;
import org.junit.Test;

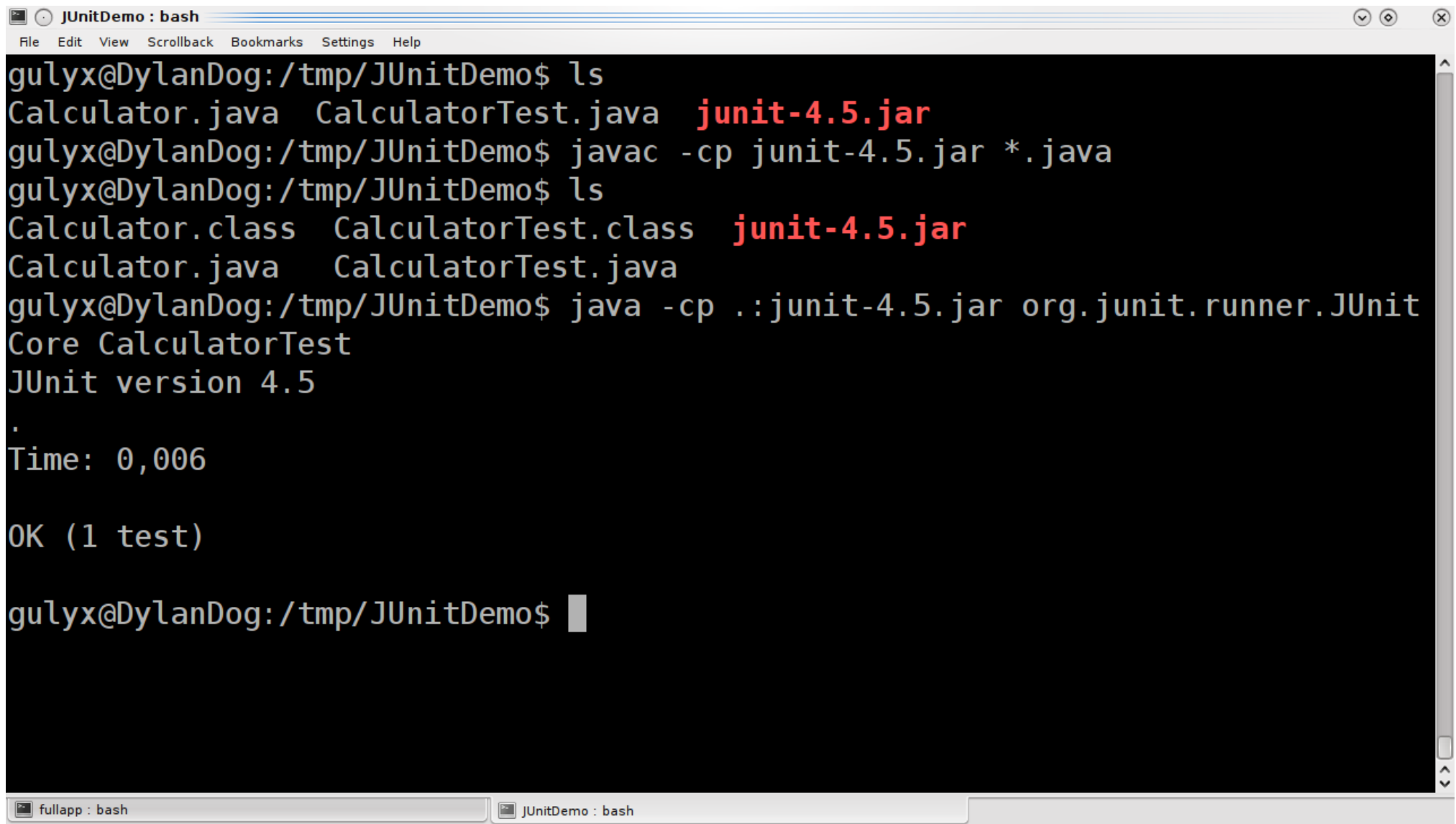
public class CalculatorTest {

    @Test
    public void testAdd() {
        Calculator calculator = new Calculator();
        double result = calculator.add(10, 50);
        assertEquals(60, result, 0);
    }
}
```



- 1 definizione del tester
- 2 annotazione che definisce un caso di test
- 3-4 implementazione del caso di test
- 5 verifica dell'esito del caso di test

la mia prima esecuzione di JUnit



```
gulyx@DylanDog:/tmp/JUnitDemo$ ls
Calculator.java  CalculatorTest.java  junit-4.5.jar
gulyx@DylanDog:/tmp/JUnitDemo$ javac -cp junit-4.5.jar *.java
gulyx@DylanDog:/tmp/JUnitDemo$ ls
Calculator.class  CalculatorTest.class  junit-4.5.jar
Calculator.java   CalculatorTest.java
gulyx@DylanDog:/tmp/JUnitDemo$ java -cp .:junit-4.5.jar org.junit.runner.JUnit4
Core CalculatorTest
JUnit version 4.5
.
Time: 0,006

OK (1 test)

gulyx@DylanDog:/tmp/JUnitDemo$
```

cosa accade sotto il cofano? – 1

```
import static org.junit.Assert.*;
import org.junit.Test;

public class CalculatorTest {

    @Test
    public void testAdd() {
        Calculator calculator = new Calculator();
        double result = calculator.add(10, 50);
        assertEquals(60, result, 0);
    }
}
```

- JUnit processa il tester ed identifica i test da eseguire
- per ogni test, JUnit carica una istanza del tester
 - riduce la possibilità di side effects
 - non riutilizzare variabili tra i vari test
- JUnit valida il caso di test attraverso la classe Assert

cosa accade sotto il cofano? – 1

```
import static org.junit.Assert.*;
import org.junit.Test;

public class CalculatorTest {

    @Test
    public void testAdd() {
        Calculator calculator = new Calculator();
        double result = calculator.add(10, 50);
        assertEquals(60, result, 0);
    }
}
```

- JUnit processa il tester ed identifica i test da eseguire
- per ogni test, JUnit carica una istanza del tester
 - riduce la possibilità di side effects
 - non riutilizzare variabili tra i vari test
- JUnit valida il caso di test attraverso la classe Assert

reflection && GOF

- quali tra i pattern GOF discussi nel corso si prestano meglio all'uso di reflection?
- perché? come?
- discutiamo le differenti opinioni/soluzioni

credits

parte dei contenuti di queste slide sono stati rielaborati prendendo spunto dalla presentazione di Manuel Mastrofini:

“Reflection”