MODEL-BASED SOFTWARE DEVELOPMENT

LECTURE VII.

MODELING GRAPHICAL LANGUAGES

Dr. Gergely Mezei
Norbert Somogyi

1

Textual modeling

Compilers, steps of language processing.
Code generation,
interpreters

2

Graphical modeling

Structure + visualization,
Blockly, UML Profile,
Metamodeling,
Semantics

3

Model processing

Model processing,
Code generation,
Graph transformation,
Model-based development

OUTLINE

TODAY'S AGENDA

I. Graphical languages / models

II. Abstract syntax based on UML

III. Blockly

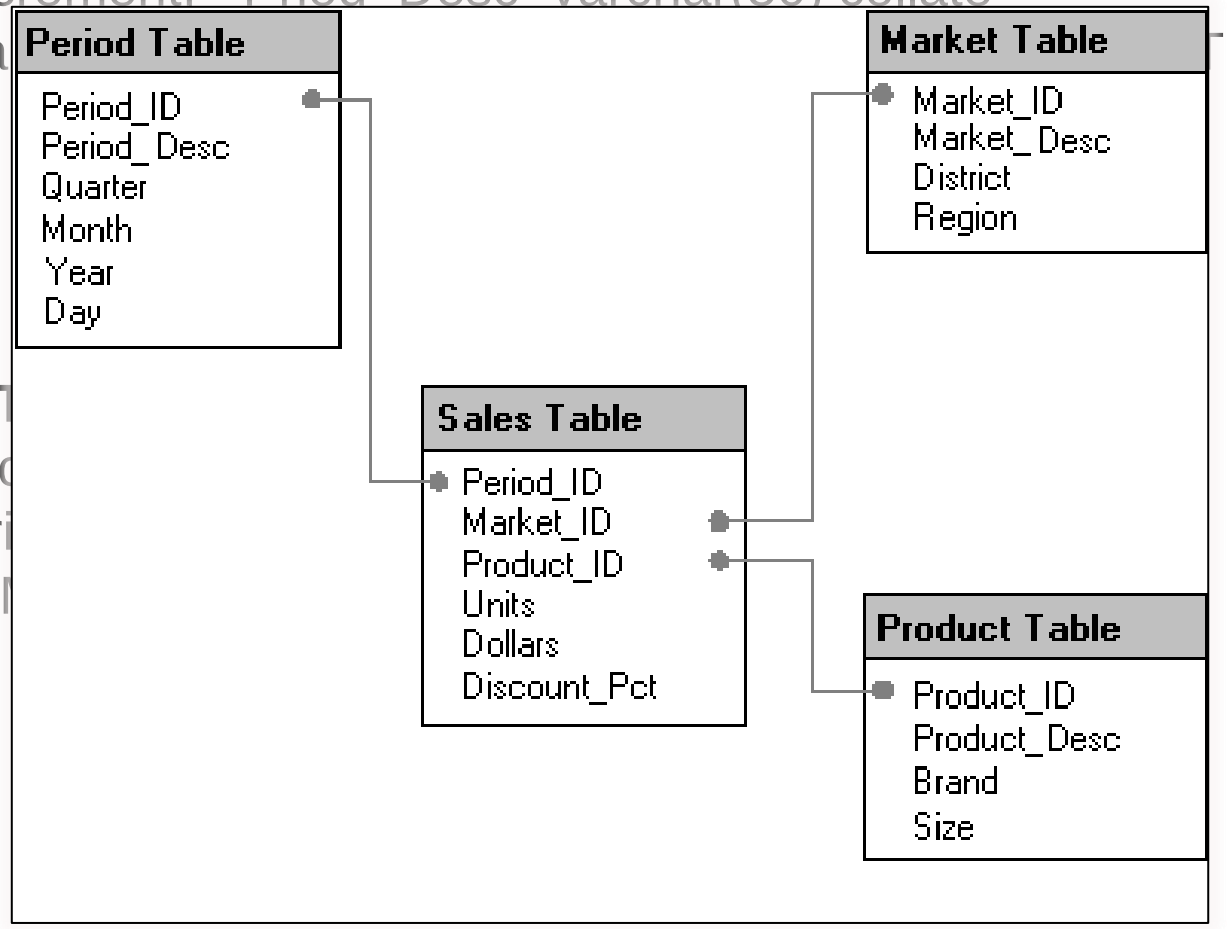
IV. Metamodeling

V. Constraints

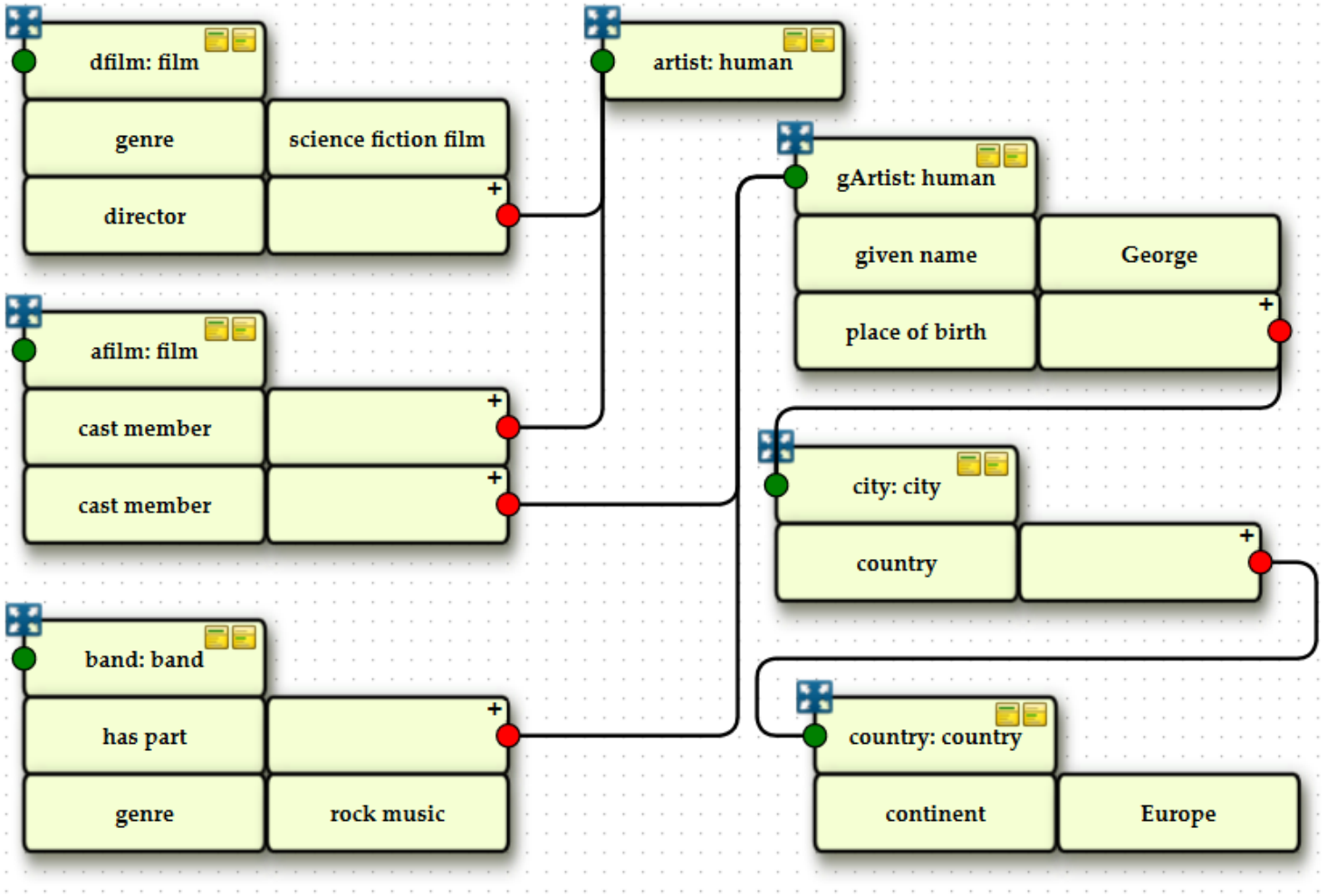
DATABASE DESIGN – SQL

CREATE TABLE IF NOT EXISTS `**Period Table**`
(`Period_ID` int(11) NOT NULL auto_increment, `Priod_Desc` varchar(50) collate latin2_hungarian_ci default NULL, `Qua
NULL, `Year` mediumint(9) NOT NULL,
(`Period_ID`));

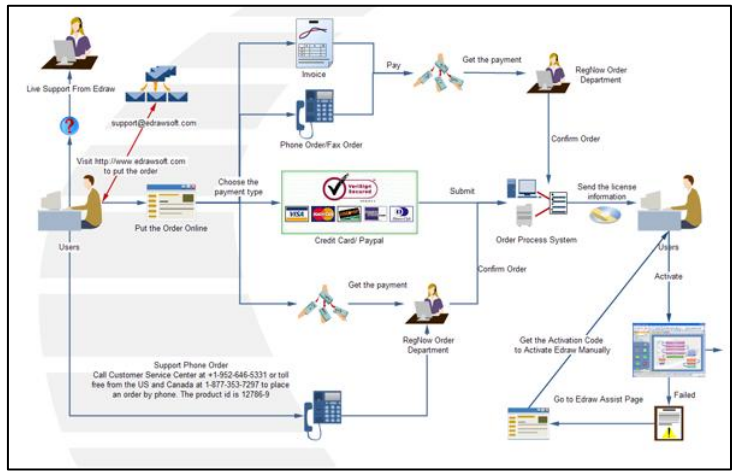
CREATE TABLE IF NOT EXISTS `**Market T**
(`Market_ID` int(11) NOT NULL auto_inc
latin2_hungarian_ci default NULL, `Distri
NULL, `Region` int(11) NOT NULL, PRI

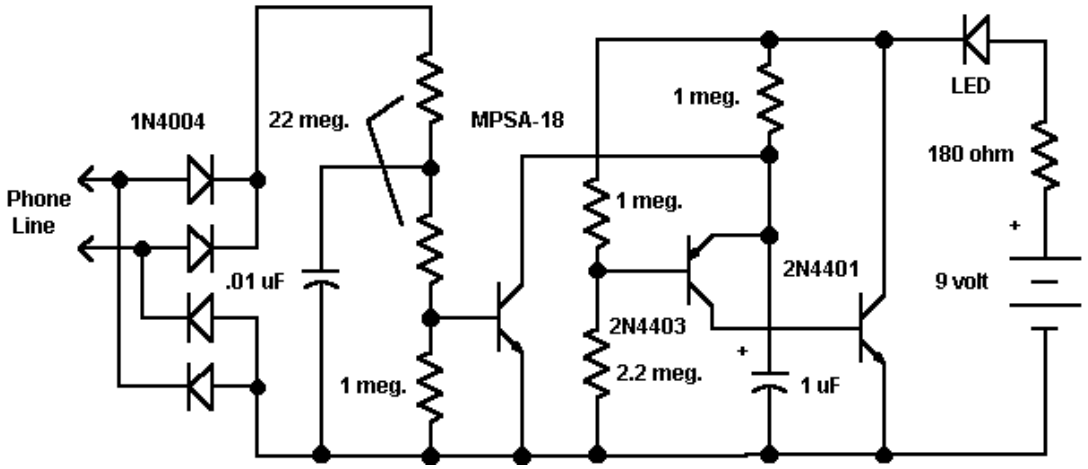


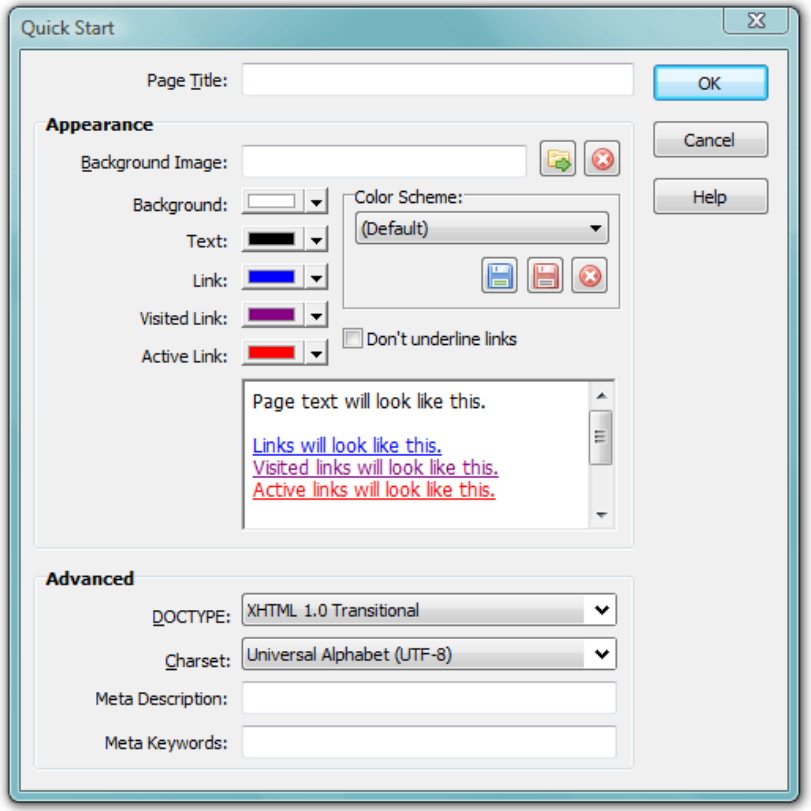
SPARQL VISUAL QUERY



FURTHER EXAMPLES







ADVANTAGES OF VISUAL MODEING

- One of the most problematic step of software development is the developer – customer communication
  - Customers are (usually) not programmers, they are not familiar with programming languages (e.g. C#, or Kotlin)!
  - However, they are familiar with the notations of their profession (domain), *which are usually visual*
  - Using these domain-specific notations, communication becomes significantly easier

TEXTUAL VS. GRAPHICAL (VISUAL) LANGUAGES

Textual languages

- Easy to write
  - Can be specified quickly
  - Complex contexts can also be described
- Difficult to understand
  - Hard to understand above a certain level of complexity
  - Syntax must be learned
- Easier to understand for developers
- Storing and version-controlling models is already solved (e.g. git)

Graphical (visual) languages

- Difficult to “write”
  - Slower, more difficult
- Easy to understand
  - Syntax is often self-describing, intuitive
  - Can be learned quickly
- Easier to understand for the “average person” (not IT-professionals)
- Storing and version-controlling models is complex, problematic (serialization)
- Sometimes the layout of model elements is part of the model

PARTS OF A DOMAIN-SPECIFIC LANGUAGE (DSL)

- What is needed to define a domain-specific language?

  - Language structure
  - Additional constraints

  } Abstract syntax

  - Visualization

  } Concrete syntax

  - The meaning of the model structure

  } Semantics

ABSTRACT – CONCRETE SYNTAX – SEMANTICS

- **Abstract syntax**:

  „Our language provides an „and” operator, which has two input parameters and an output result. All of these are of type bool.”

- **Concrete syntax**:

  „The „and” operator is denoted by „&&” ”

- **Semantics**:

  The “and” operator should return true if and only if both its input parameters are logical true.

TODAY'S AGENDA

I. Graphical languages / models

II. Abstract syntax based on UML

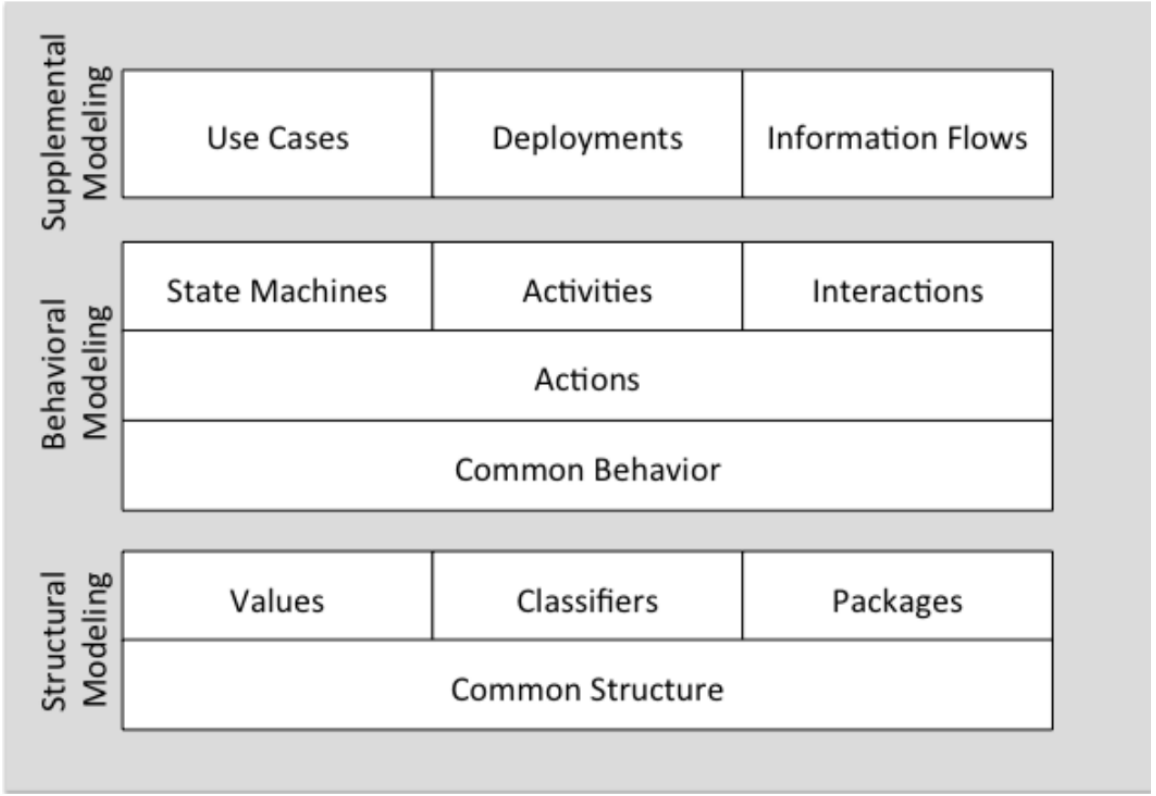
III. Blockly

IV. Metamodeling

V. Constraints

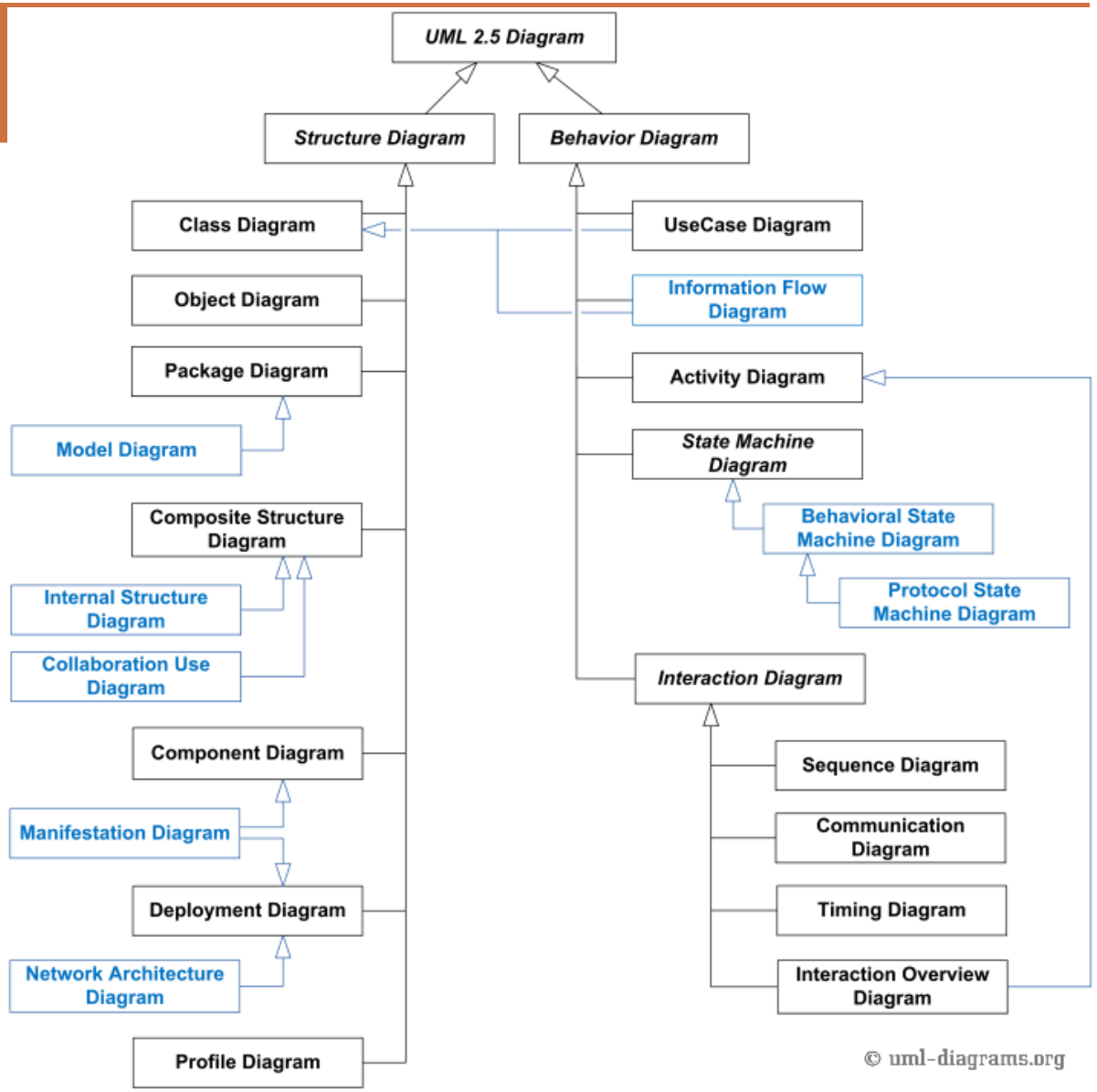
PROPERTIES OF UML

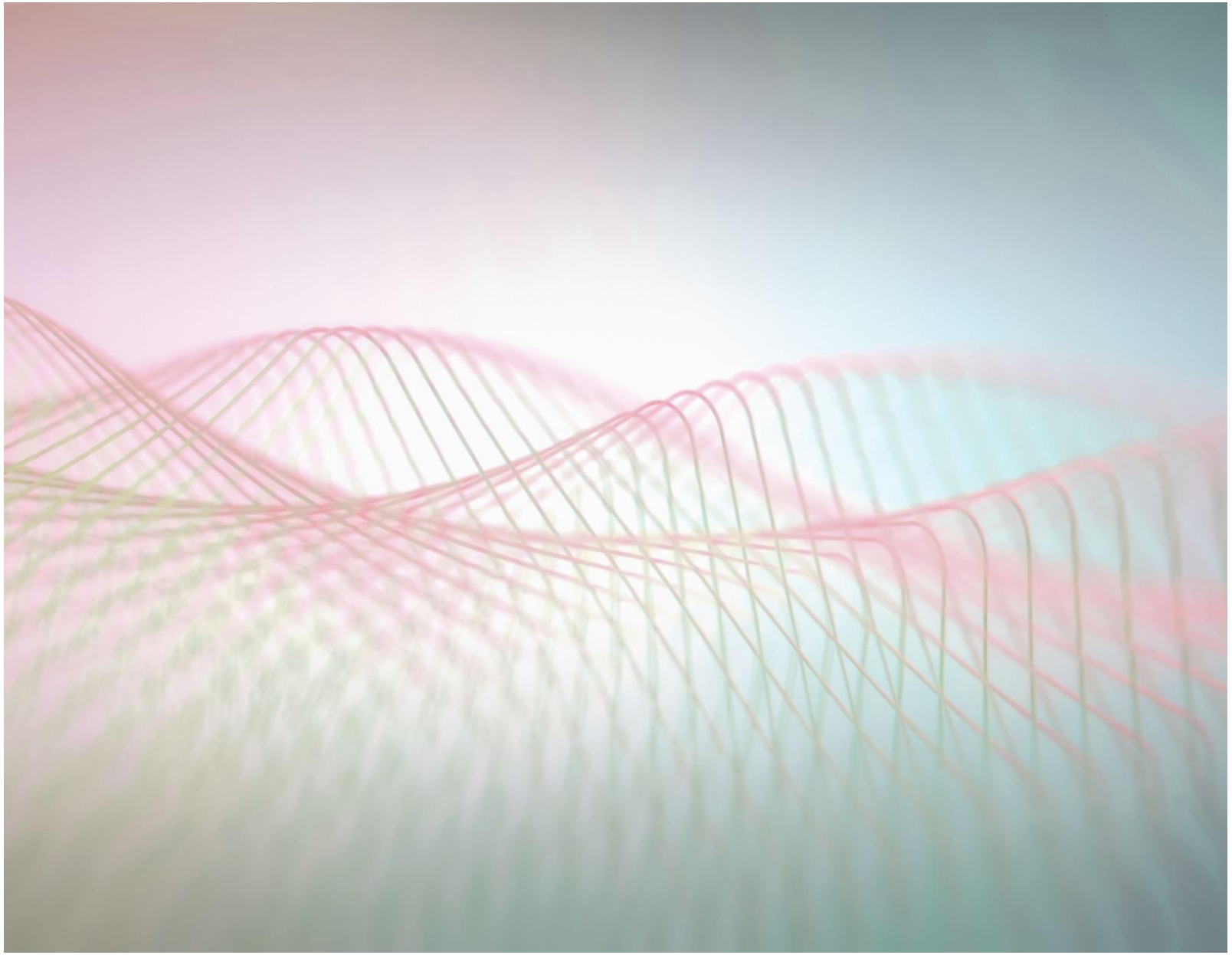
- Visual modeling language
- Combines the advantages of the methodologies developed so far
- Programming language independent
- Extensible (mainly: stereotypes)
- Standardized (currently version 2.5.1)



UML LANGUAGES

- UML offers a powerful way to use models in software design
- 14 diagram types (UML 2.5.1)
- How can it be better customised?





UML Profile

UML PROFILE

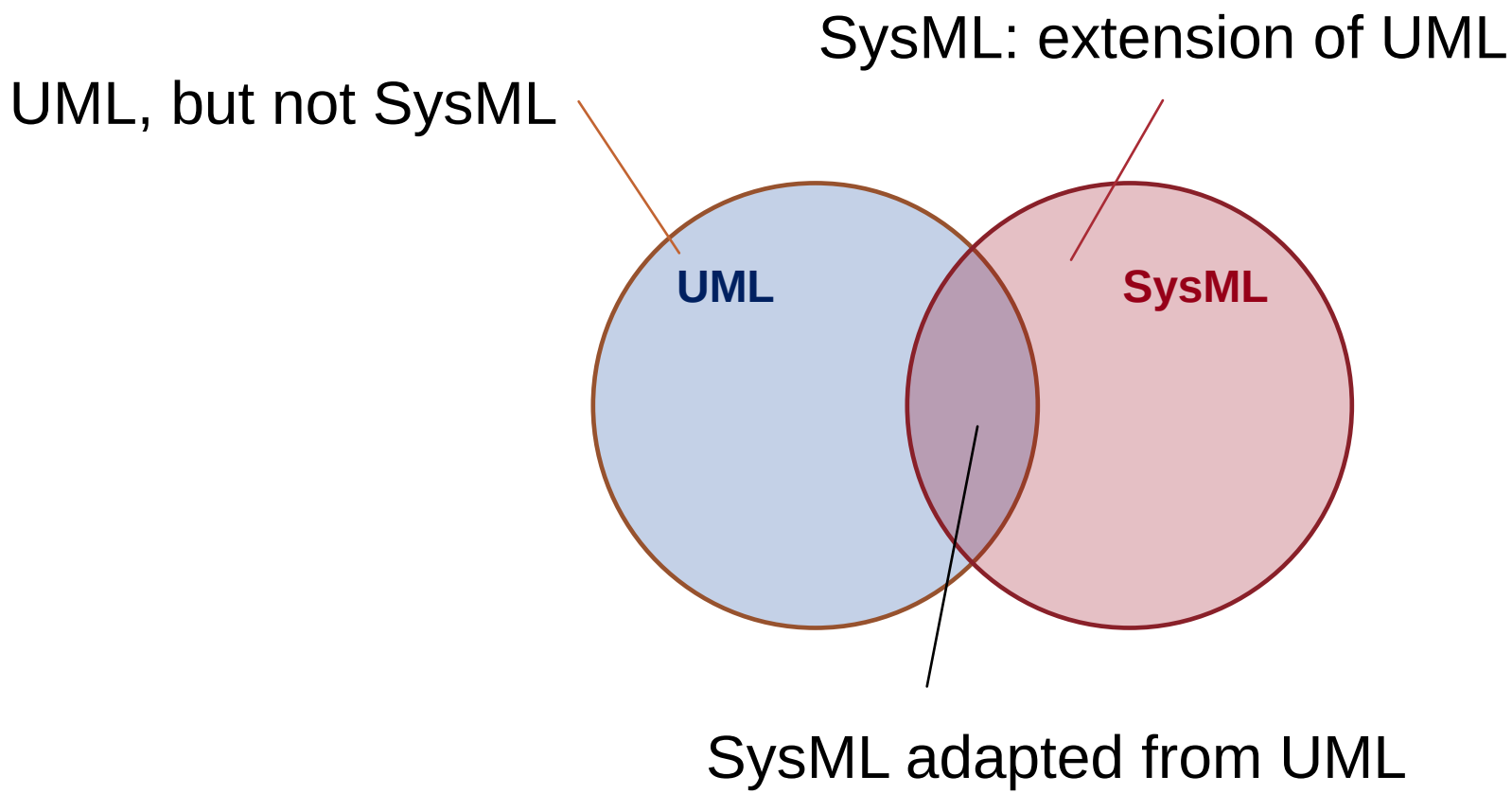
- It may be necessary to specialize the general languages offered by UML
  - E.g. Telecommunication models, embedded systems
  - How can domain-specific languages be expressed in UML?
- OMG solution: UML Profile
  - Selects a subset of UML models, provides "well-formedness" in the selected domain
  - Describes the specialization of general elements using stereotypes, tags and constraints
  - Does not contradict the original specification

UML PROFILE EXAMPLES

- Modeling and Analysis of Real Time and Embedded systems (MARTE)
- Service oriented architecture Modeling Language (SoaML)
- UML Profile for Advanced and Integrated Telecommunication Services (TelcoML)
- UML Testing Profile (UTP)
- UML Profile for Voice
- SysML

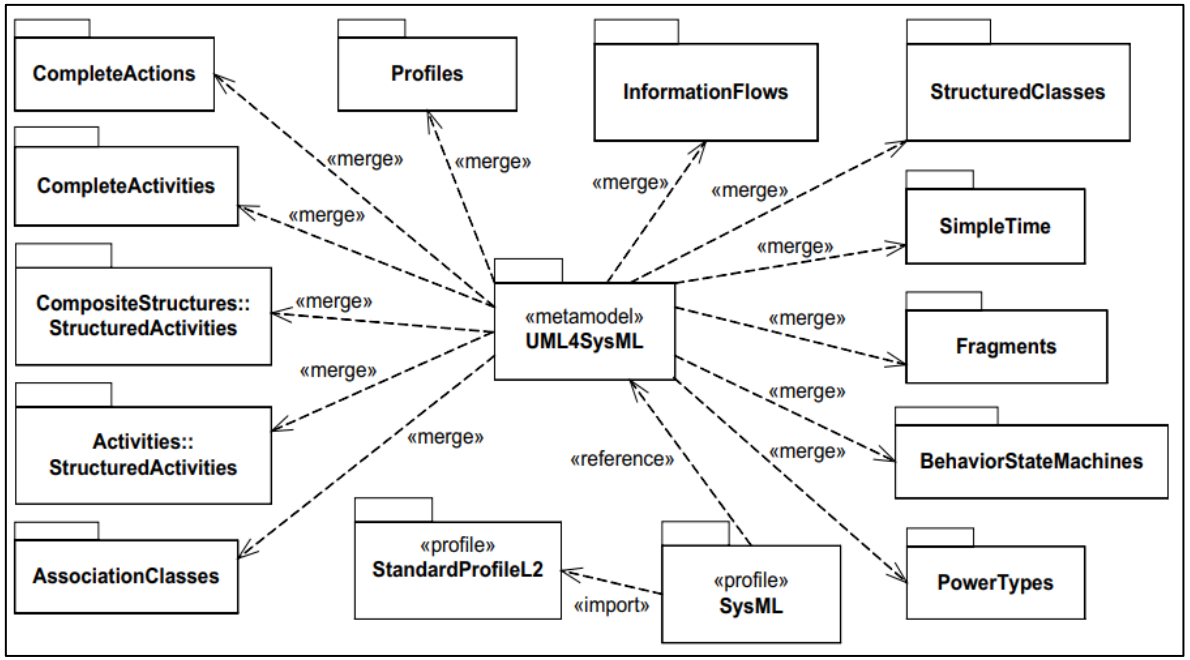
SYSML

- General modelling, for system design
  - Specification
  - Analysis
  - Design
  - Verification
  - Validation



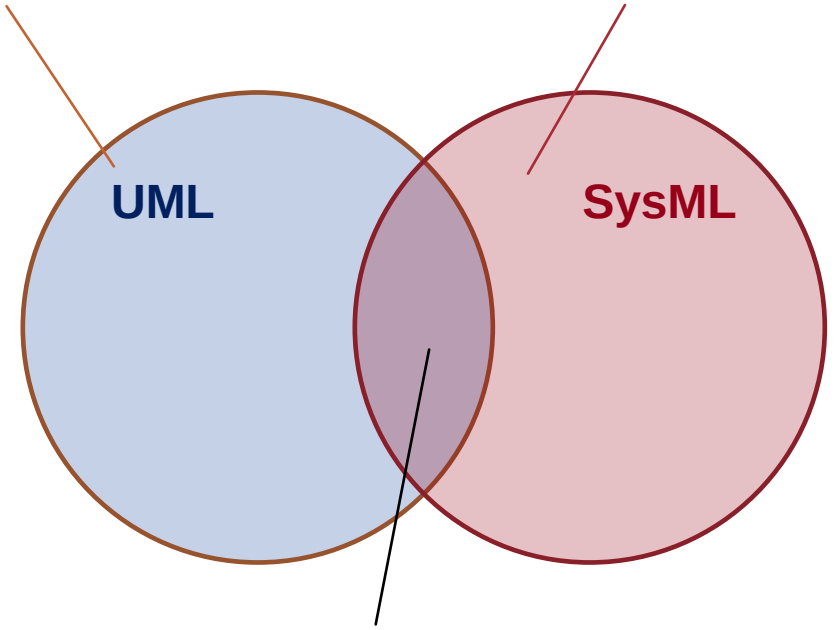
SYSML

- Omits 7 UML diagrams



UML, but not SysML

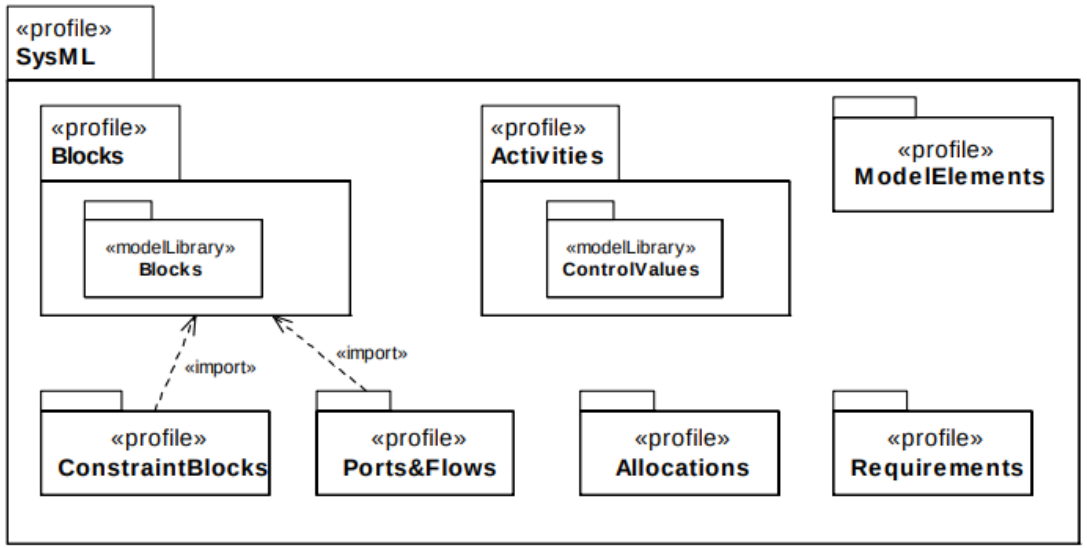
SysML: extension of UML

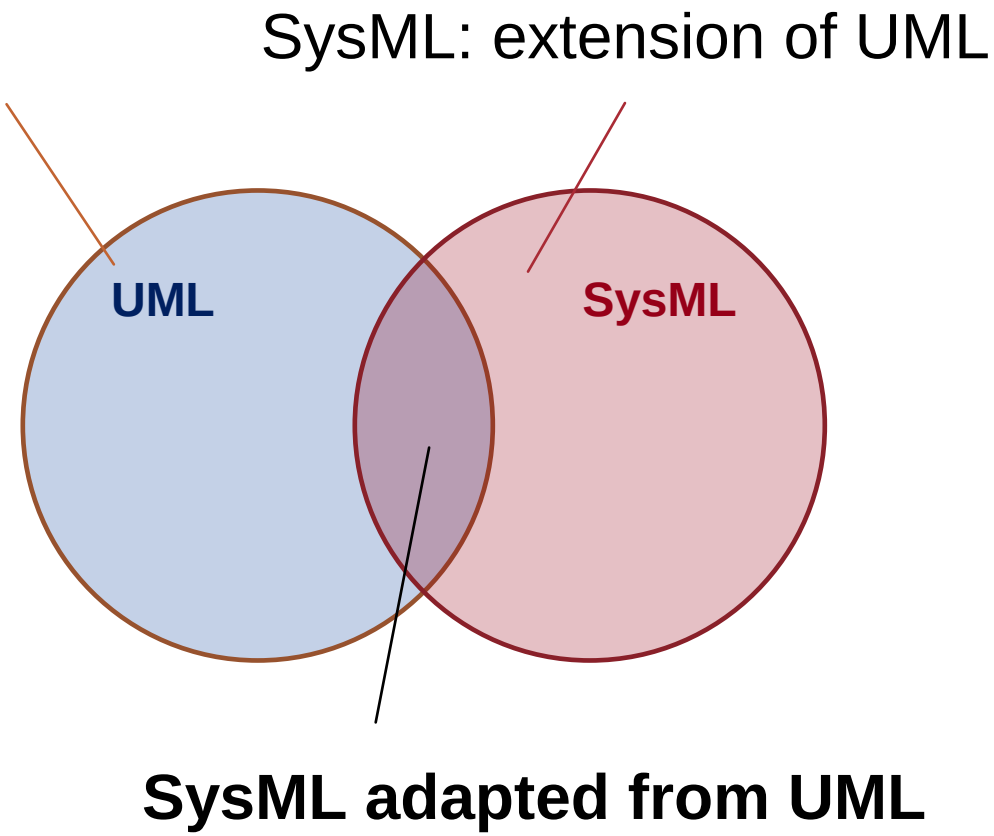


SysML adapted from UML

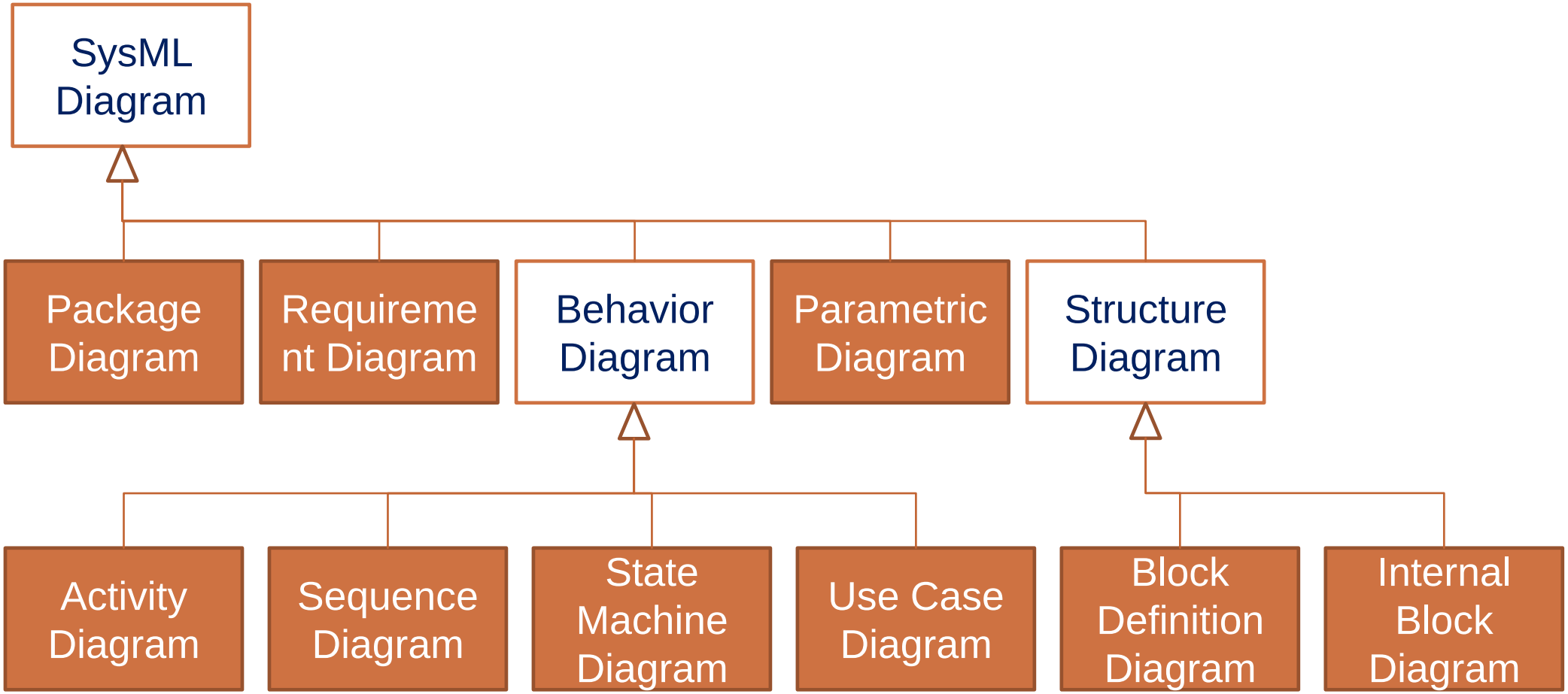
SYSML

- Adds new diagrams





SYSML DIAGRAM TYPES



TODAY'S AGENDA

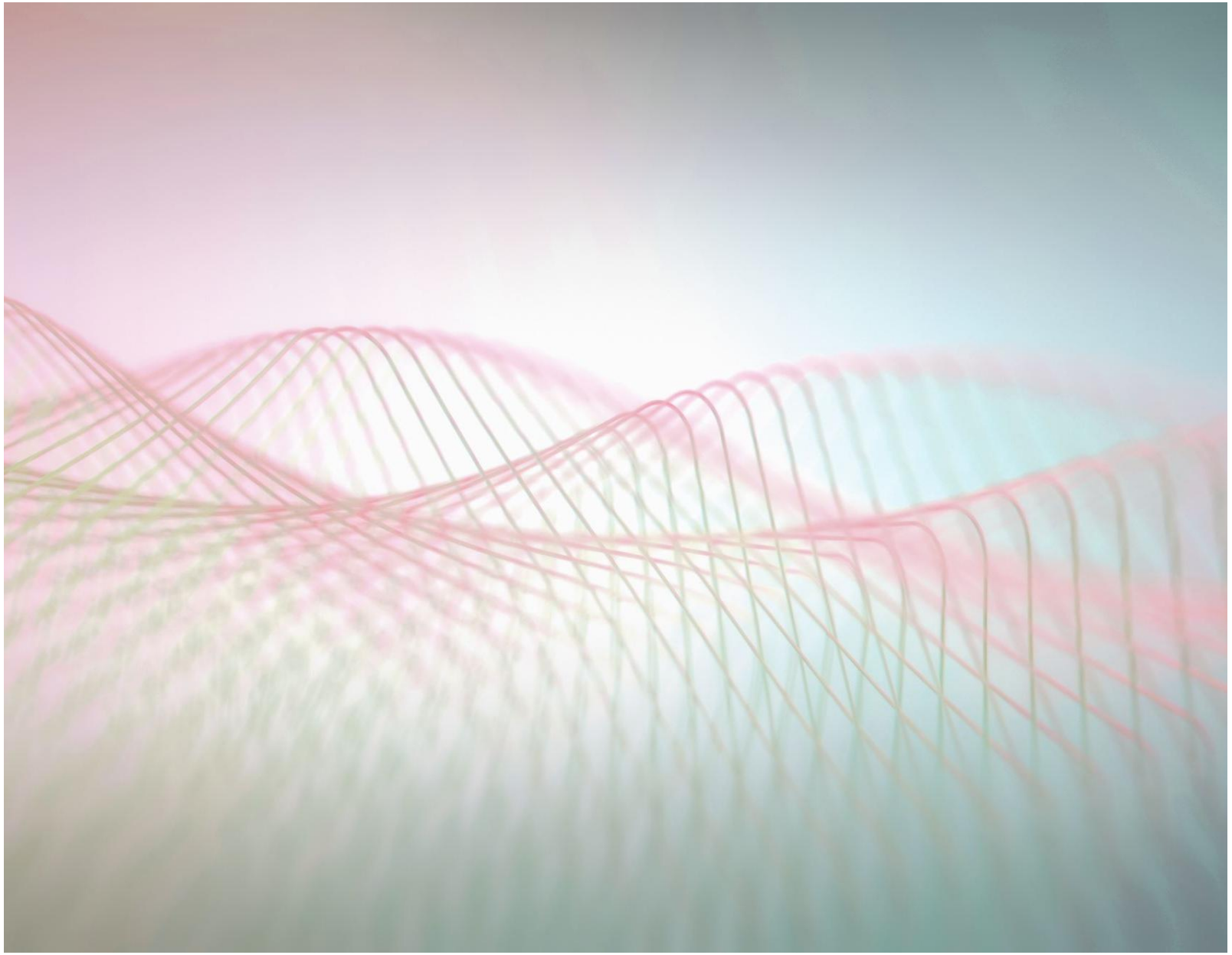
I. Graphical languages / models

II. Abstract syntax based on UML

III. Blockly

IV. Metamodeling

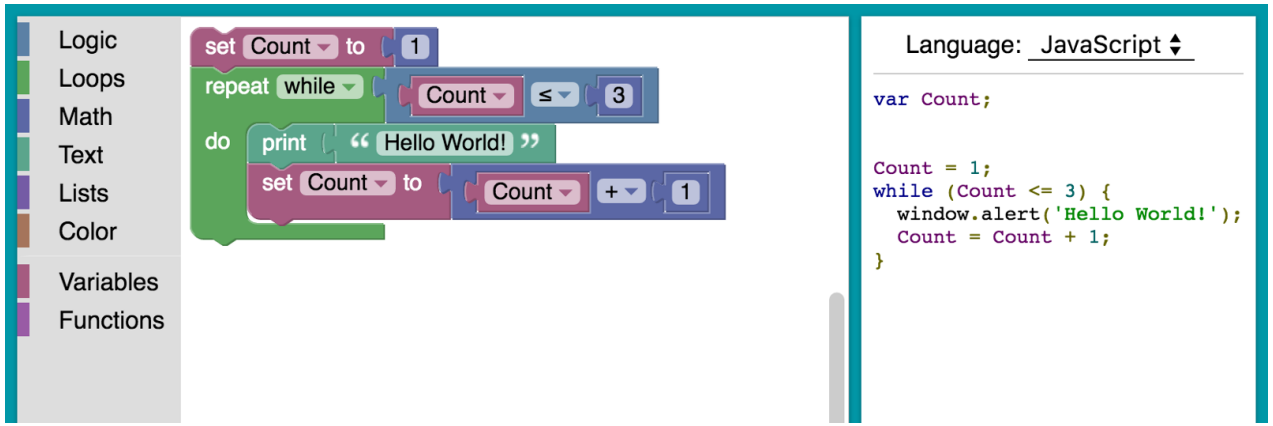
V. Constraints



Blockly

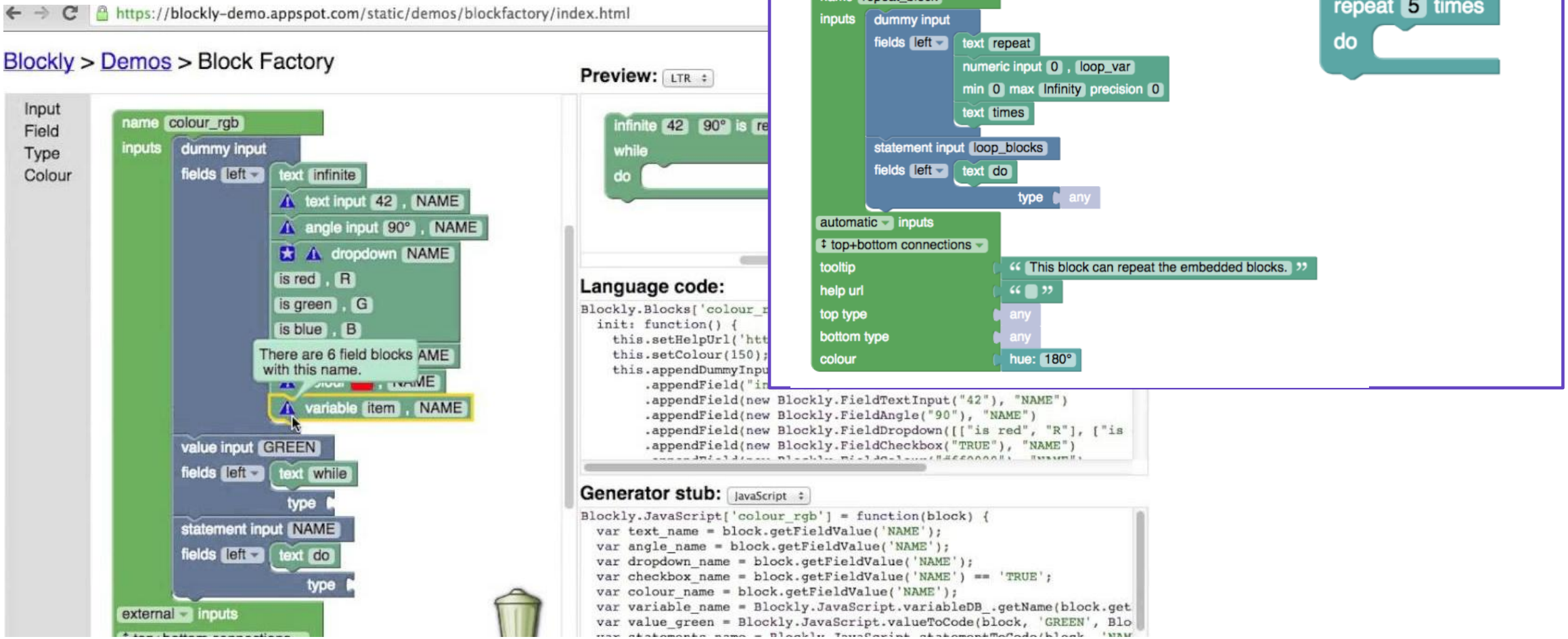
BLOCKLY

- Nested blocks
- Simple, visual programming language
- General, can be customized
- Projection-based editor
- Template-based code generation



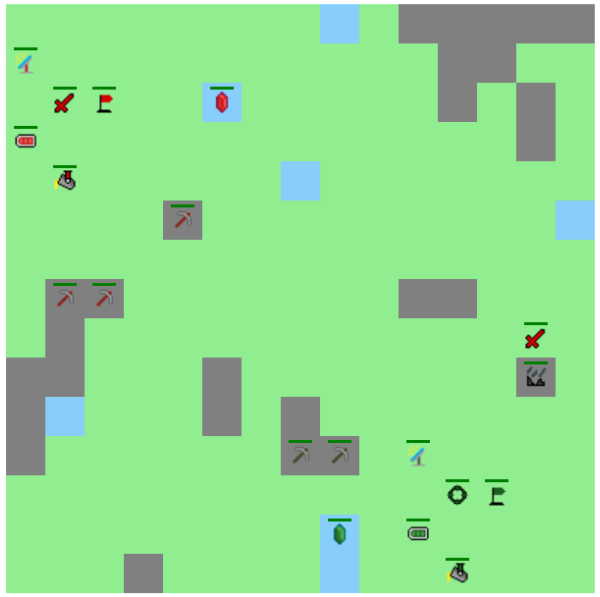
BLOCKLY

- Block factory – custom blocks

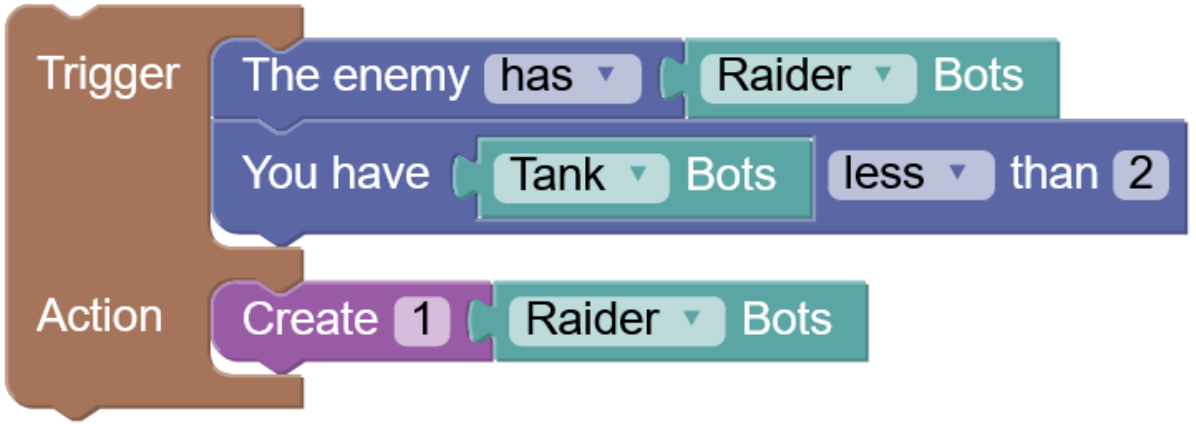


BLOCKLY EXAMPLE – STRATEGY GAME AI LANGUAGE

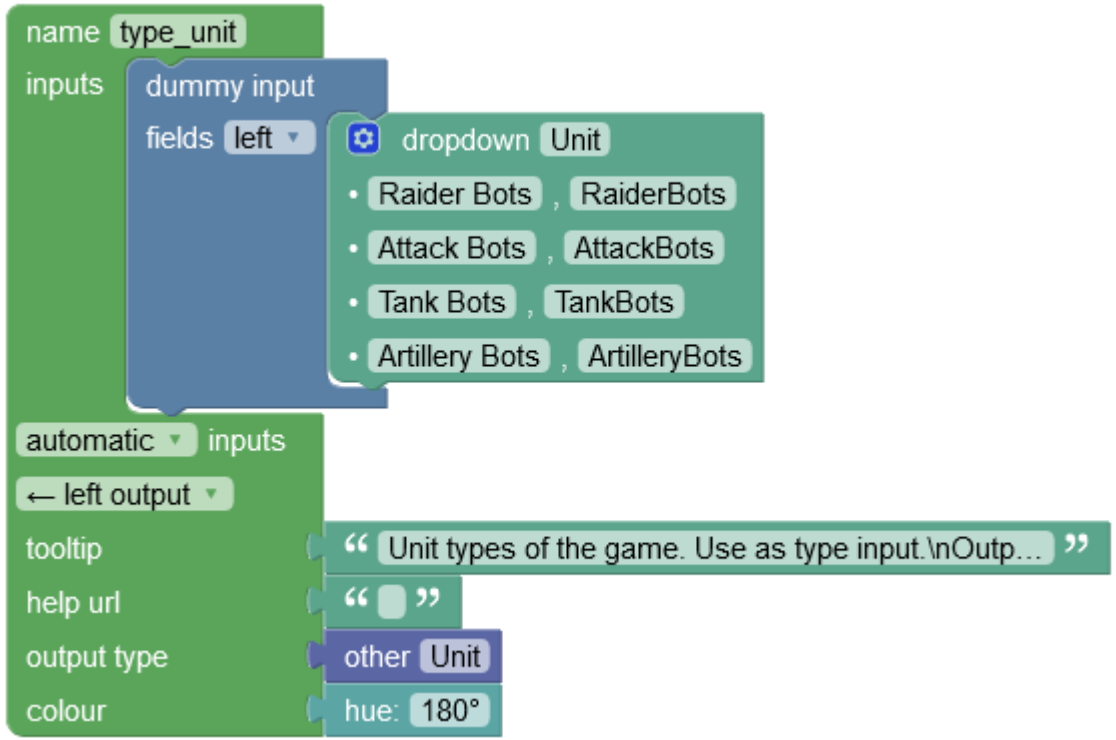
Turn	Player	Steel	RoboSteel	Energy	Crystals	Energy Cores	Credits
13	Player	20	5	60	16	7	85
13	Server AI	10	5	65	6	7	120

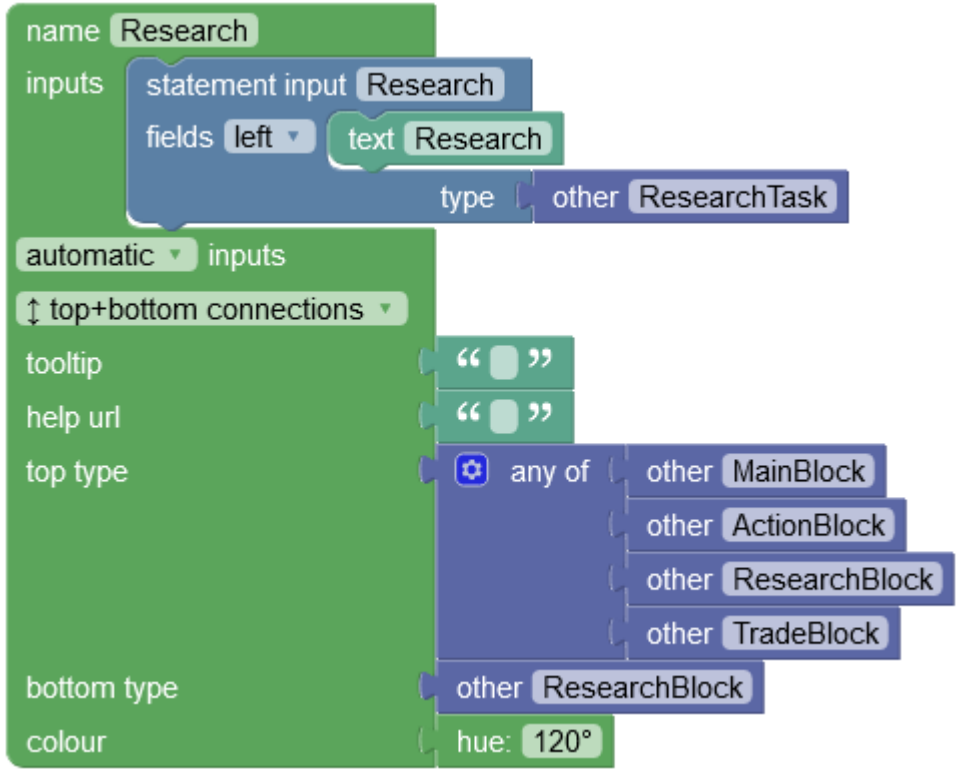


Player
artilleryBotUpgrades
attackBotUpgrades
raiderBotUpgrades
tankBotUpgrades
Server AI
artilleryBotUpgrades
attackBotUpgrades
raiderBotUpgrades
tankBotUpgrades

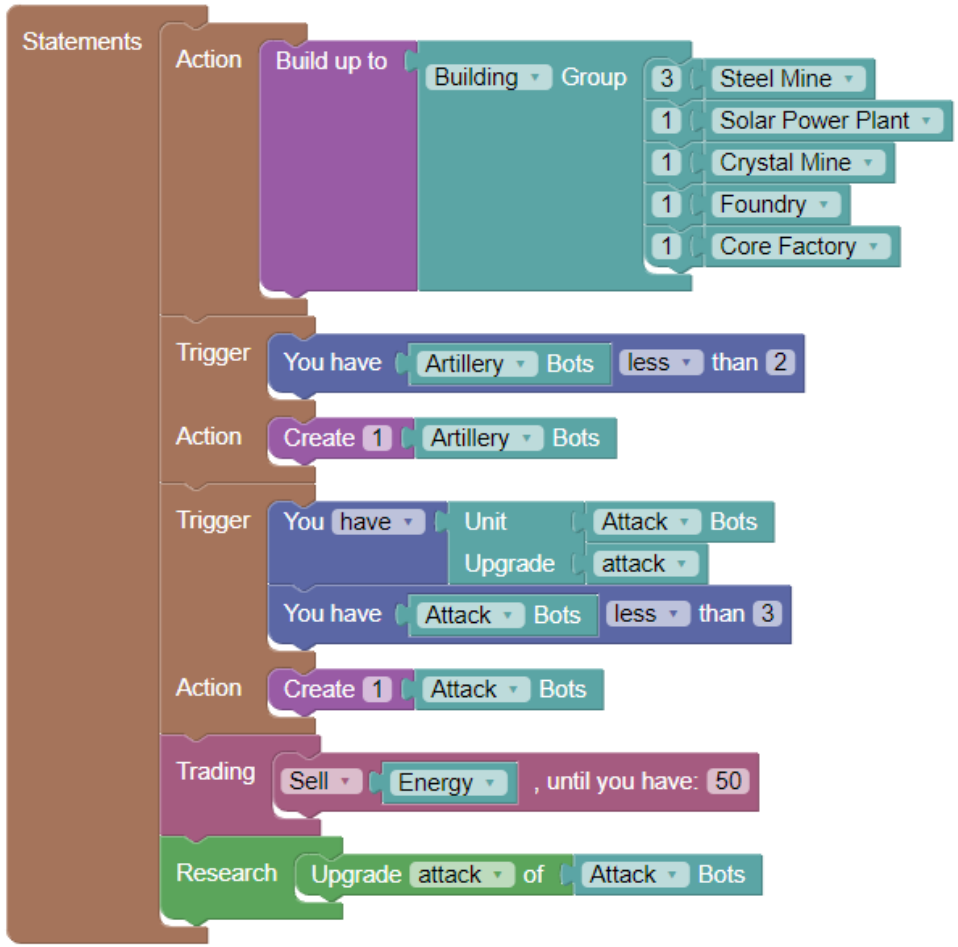


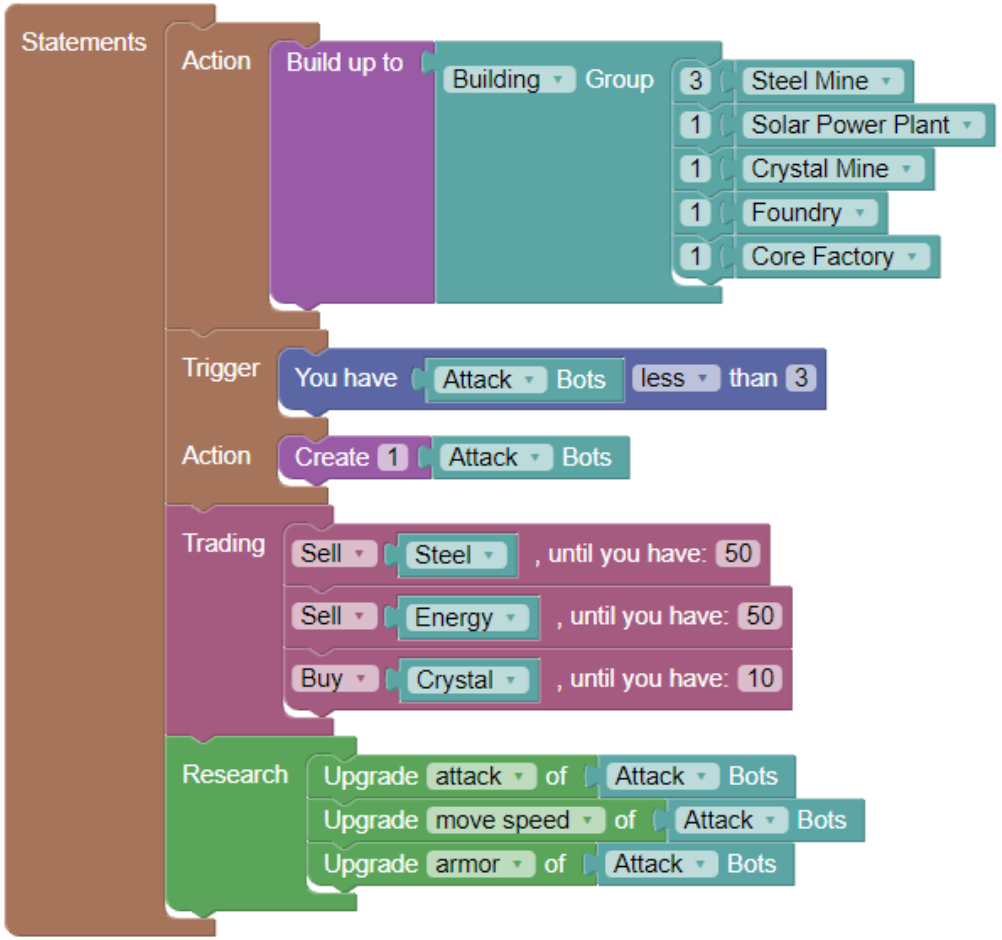
BLOCKLY EXAMPLE – STRATEGY GAME AI LANGUAGE





BLOCKLY EXAMPLE – STRATEGY GAME AI LANGUAGE





TODAY'S AGENDA

I. Graphical languages / models

II. Abstract syntax based on UML

III. Blockly

IV. Metamodeling

V. Constraints

METAMODELING

- Metamodel: defines the basic modelling elements, their relationships and structural constraints
  - Model elements
  - Relations between the elements
  - Attributes (of model elements and relations)
- Extensible: constraints and rules

METAMODELING VS…

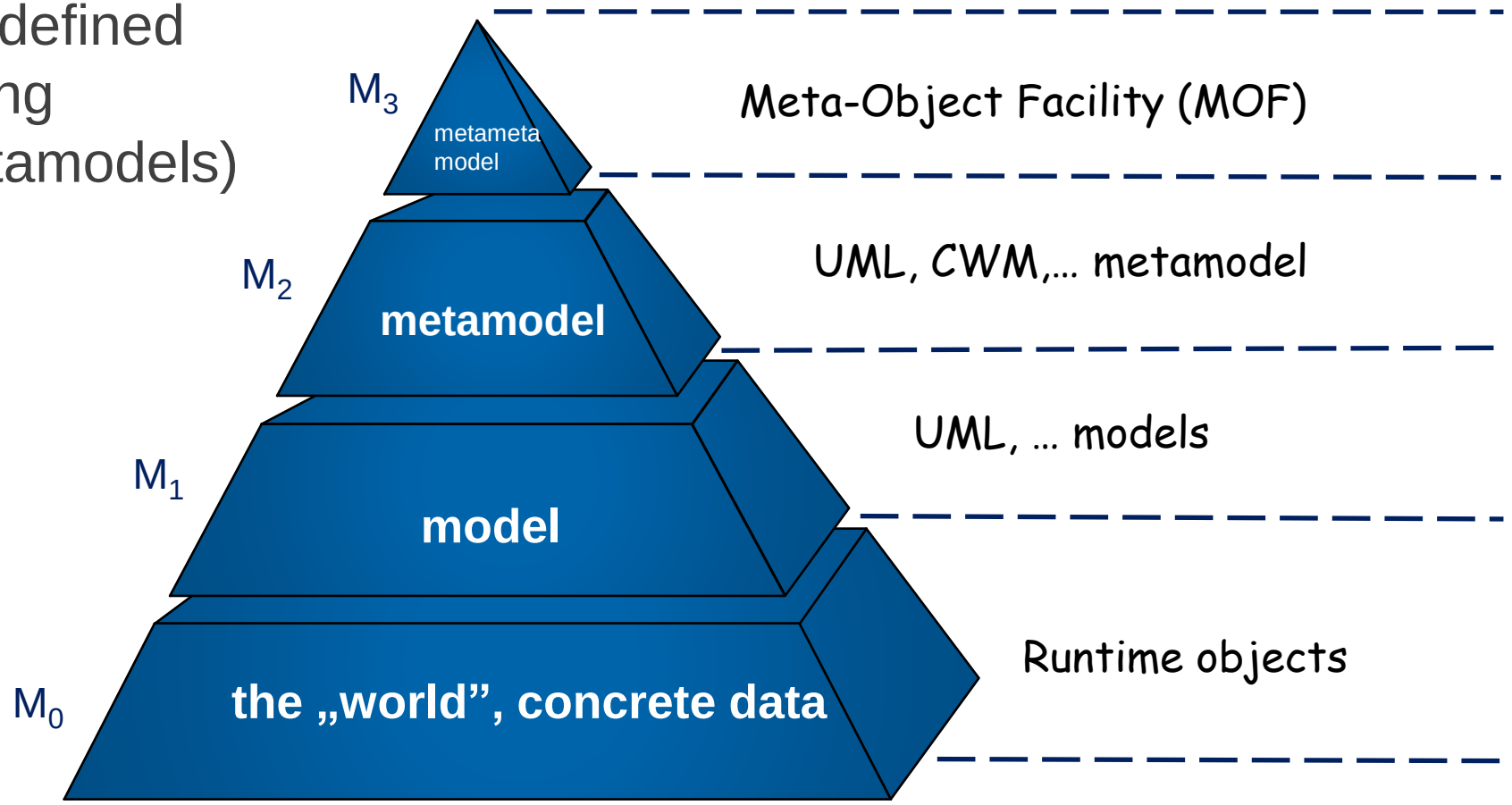
- UML profile
  - Bound by the rules of UML
  - Less flexible / customizable
- Blockly
  - Restricted controls on template fields
  - Defining Relations and repeated information (function calls)
- Ad-hoc custom solution
  - Not formal / standardized → automatic processing can be difficult
  - Lack of common language, interoperability

METAMODELING LANGUAGE

- The abstract syntax of models is defined by metamodels
- How to describe the abstract syntax of *metamodels*?
- Metamodeling language
  - A (special) domain-specific language
  - Can be used to define metamodels
  - Define the *possible* abstract syntax of modelling languages

META-OBJECT FACILITY (MOF)

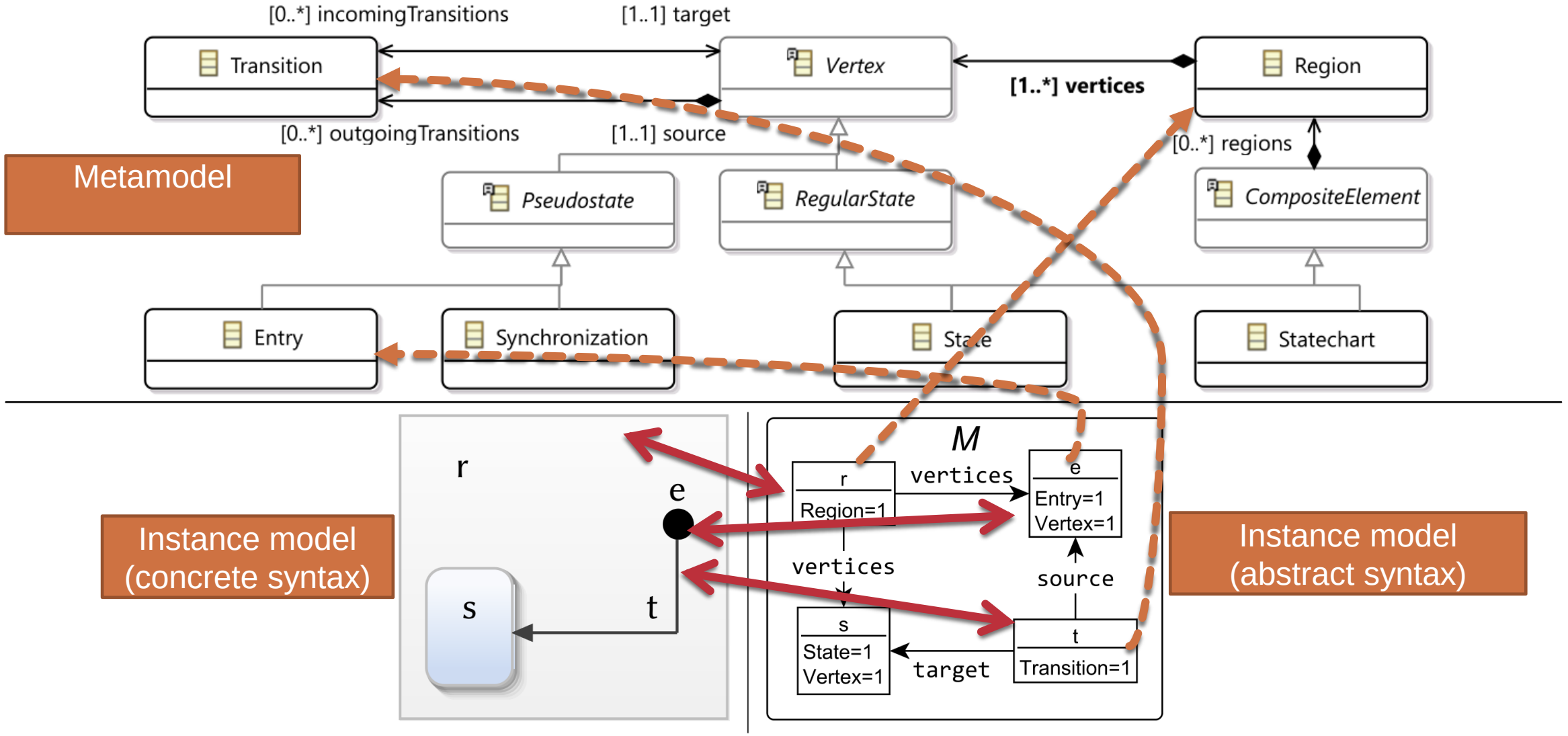
- Object Management Group (OMG) standard
- Semantically well-defined model for describing type systems (metamodels)



MOF – 4 levels

Level	Description	Examples	Who creates it?
Metameta-model	Metamodel architecture to create Metamodeling languages.	MetaClass, MetaAttribute, MetaOperation	Researchers, MOF developers
Metamodel	Instance of the metameta-model. To describe domain-specific and modeling languages.	Class, Attribute, Operation	Standard developers, DSL developers
Model	Instance of the metamodel, concrete models of the domain.	Person class, Group class	Experts, users
Objects, data	Instance of the model, concrete model with actual, concrete data	„John Doe”, ”Marketing department group”	Runtime objects

YAKINDU EXAMPLE



MOF VARIANTS

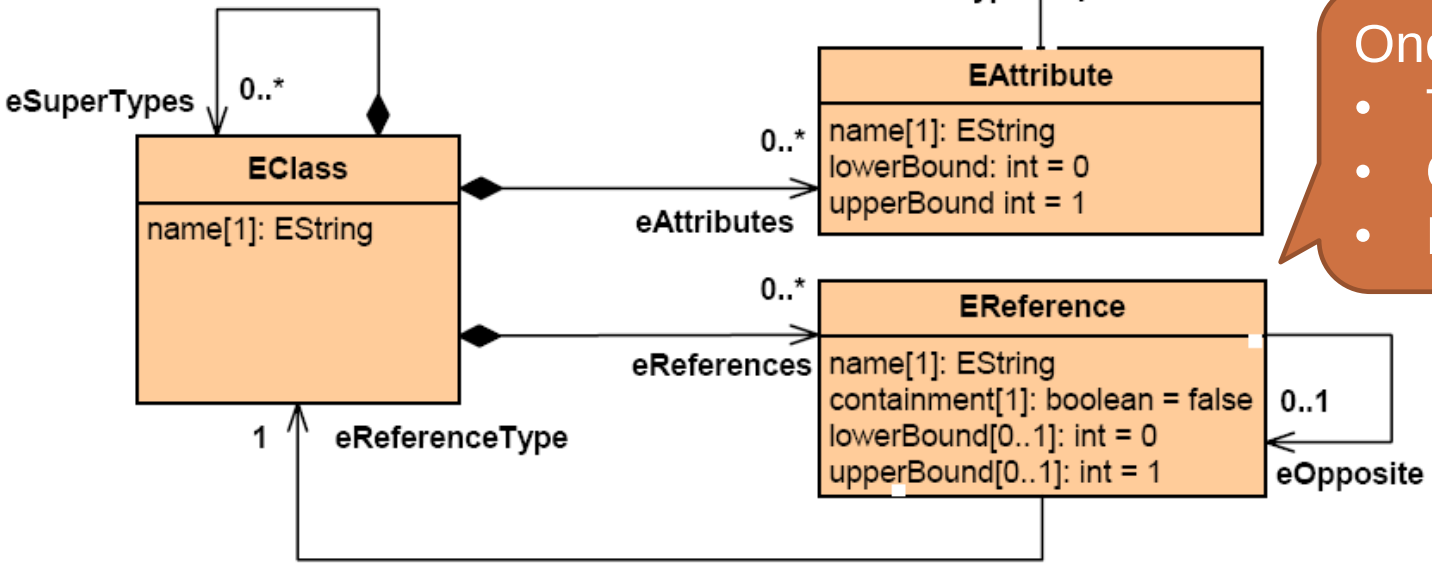
- EMOF (Essential MOF)
  - Basic functionalities (related to OO and XML)
  - Goal: map MOF models to JMI and XMI format
  - For simple metamodels
  - Supports customization
  - ECore
- CMOF (Complete MOF)
  - „Complete version” (UML 2.0 extensions)
  - For defining languages similar to UML 2.0

ECORE BASICS

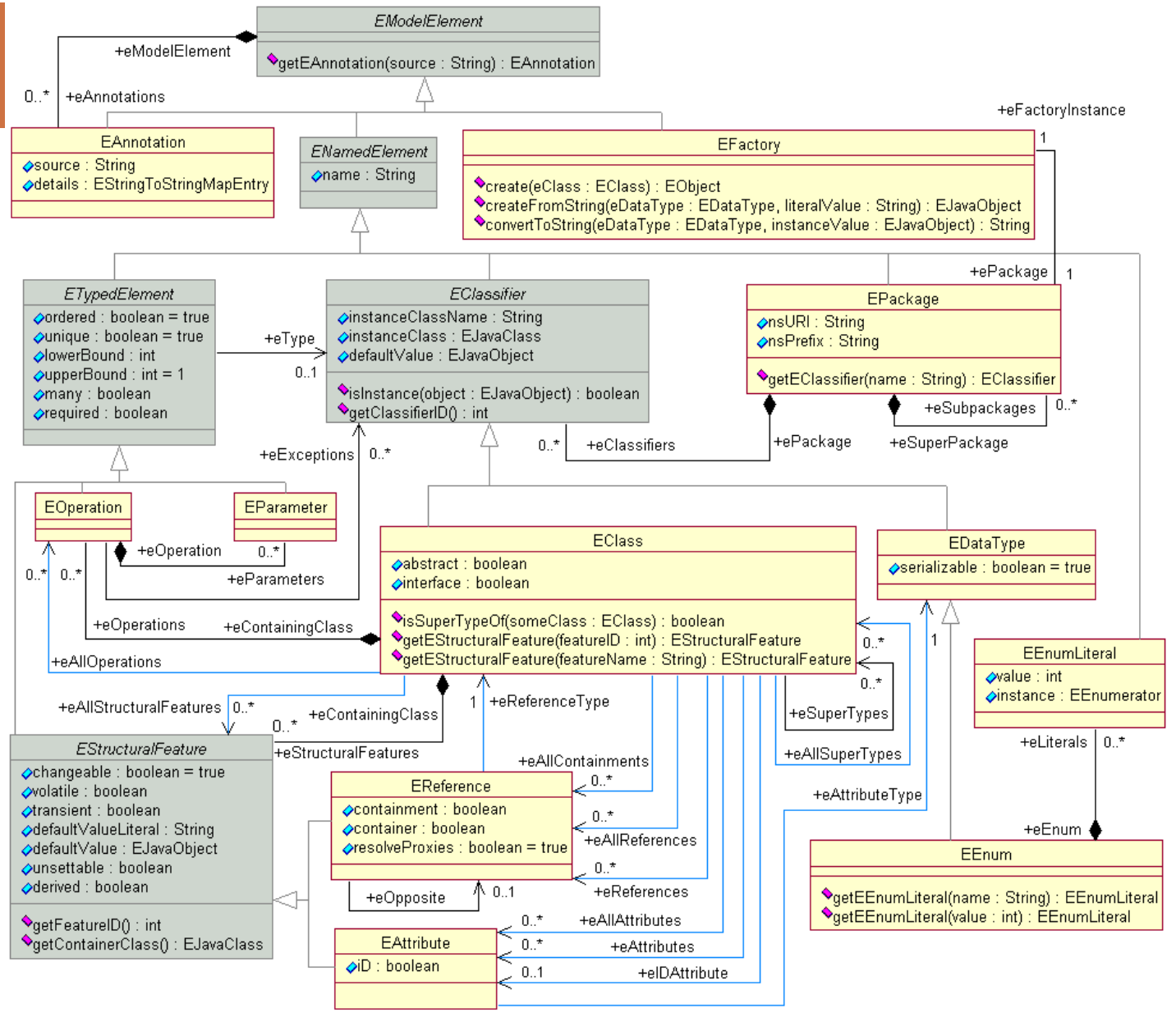
Concept
- super-concepts
- associations
- attributes

Typed attribute

One-way binary relation
- Type
- Optional inverse
- Multiplicity

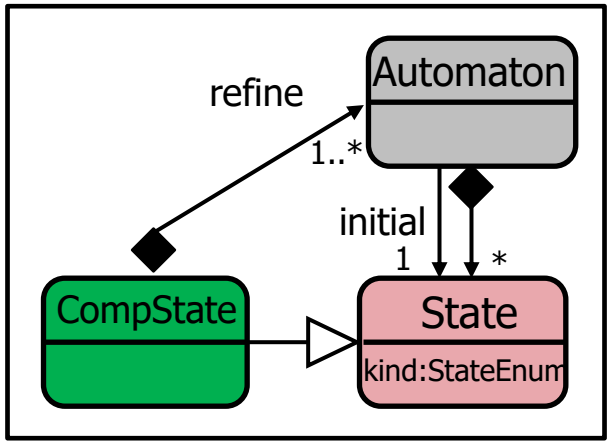


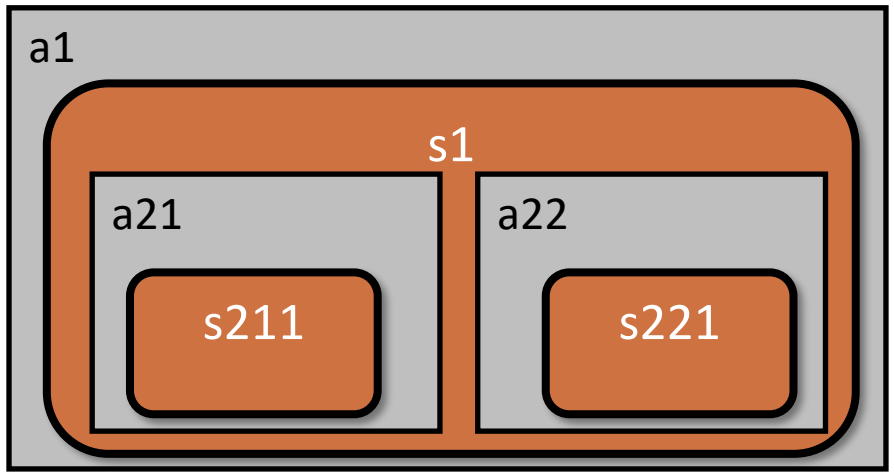
Ecore – EMOF*

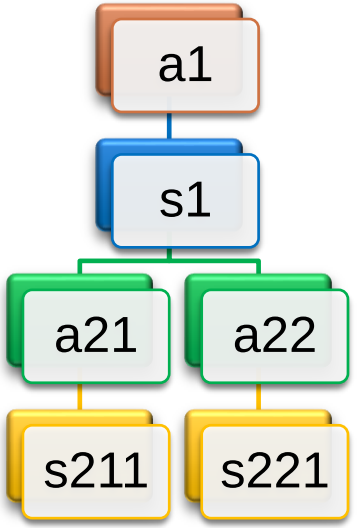

EModelElement
getEAnnotation(source : String) : EAnnotation
+eModelElement
0..*
+eAnnotations
EAnnotation
source : String
details : EStringToStringMapEntry
ENamedElement
name : String
EFactory
create(eClass : EClass) : EObject
createFromString(eDataType : EDataType, literalValue : String) : EJavaObject
convertToString(eDataType : EDataType, instanceValue : EJavaObject) : String
+eFactoryInstance
1
ETypedElement
ordered : boolean = true
unique : boolean = true
lowerBound : int
upperBound : int = 1
many : boolean
required : boolean
+eType
0..1
EClassifier
instanceClassName : String
instanceClass : EJavaClass
defaultValue : EJavaObject
isInstance(object : EJavaObject) : boolean
getClassifierID() : int
+ePackage
1
EPackage
nsURI : String
nsPrefix : String
getEClassifier(name : String) : EClassifier
+eSubpackages
0..*
+eSuperPackage
+eClassifiers
0..*
+ePackage
+eExceptions
0..*
EOperation
EParameter
+eOperation
0..*
+eParameters
0..*
0..*
+eOperations
+eContainingClass
+eAllOperations
EClass
abstract : boolean
interface : boolean
isSuperTypeOf(someClass : EClass) : boolean
getEStructuralFeature(featureID : int) : EStructuralFeature
getEStructuralFeature(featureName : String) : EStructuralFeature
EDataType
serializable : boolean = true
EEnumLiteral
value : int
instance : EEnumerator
1
+eLiterals
0..*
+eEnum
EEnum
getEEnumLiteral(name : String) : EEnumLiteral
getEEnumLiteral(value : int) : EEnumLiteral
+eAllStructuralFeatures
0..*
0..*
+eContainingClass
+eStructuralFeatures
1
+eReferenceType
0..*
+eSuperTypes
+eAllSuperTypes
+eAttributeType
EStructuralFeature
changeable : boolean = true
volatile : boolean
transient : boolean
defaultValueLiteral : String
defaultValue : EJavaObject
unsettable : boolean
derived : boolean
getFeatureID() : int
getContainerClass() : EJavaClass
+eAllContainments
0..*
EReference
containment : boolean
container : boolean
resolveProxies : boolean = true
0..*
+eAllReferences
0..*
+eReferences
+eOpposite
0..1
0..*
+eAllAttributes
EAttribute
iD : boolean
0..*
+eAttributes
0..1
+eIDAttribute

CONTAINMENT

- Every model elements has exactly one container
- Containment (as a relation) is modeled as well
  - > Special edge between meta elements
  - > With multiplicity rules
- Cyclic containment is prohibited
  - > But possible in the metamodel, on the level of types







INHERITANCE VS INSTANTIATION

1. Jack is a bulldog
2. A bulldog is a dog
3. A dog is an animal
4. A bulldog is a (dog)breed
5. A dog is a species

✓ 1+2 = Jack is a dog

✓ 1+2+3 = Jack is an animal

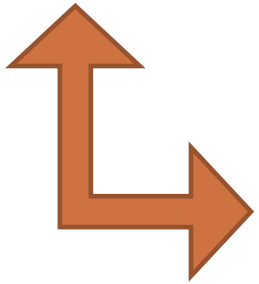
! 1+4 = Jack is ***not*** a (dog)breed

! 2+5 = Bulldog is ***not*** a species

- Inheritance (SupertypeOf): subset, transitive
- Instantiation (InstanceOf): ‘template’ fill-in, not transitive

SERIALIZING METADATA - XMI

- Transfer of models in heterogeneous environmentsű
- **X**ML **M**etadata **I**nterchange (XMI)
  - > OMG standard
  - > Part: MOF → XML mapping



```
<Classifiers xsi:type="Class" name="Department">
   <StructuralFeatures xsi:type="Attribute" name="id" Type="String"/>
   <StructuralFeatures xsi:type="Reference" name="member"
upperBound="-1"
       Type="#//Employee" containment="true"/>
 </Classifiers>
 <Classifiers xsi:type="Class" name="Employee">
   <StructuralFeatures xsi:type="Attribute" name="name" Type="String"/>
 </Classifiers>
```

TODAY'S AGENDA

I. Graphical languages / models

II. Abstract syntax based on UML

III. Blockly

IV. Metamodeling

V. Constraints

PARTS OF A DOMAIN-SPECIFIC LANGUAGE (DSL)

- What is needed to define a domain-specific language?

  - Language structure
  - Additional constraints

  } Abstract syntax

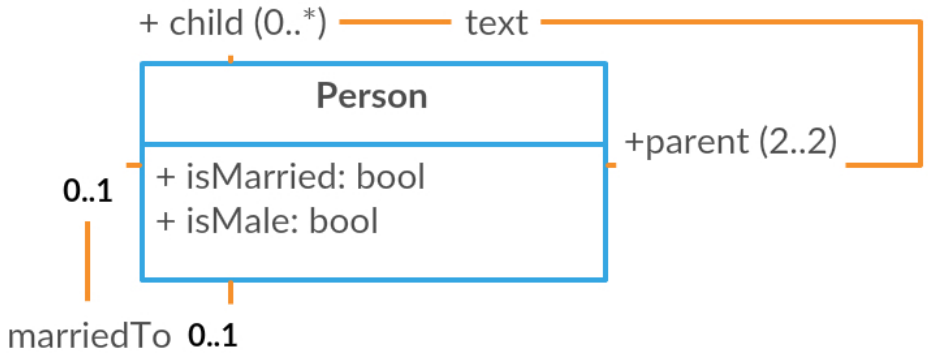
  - Visualization

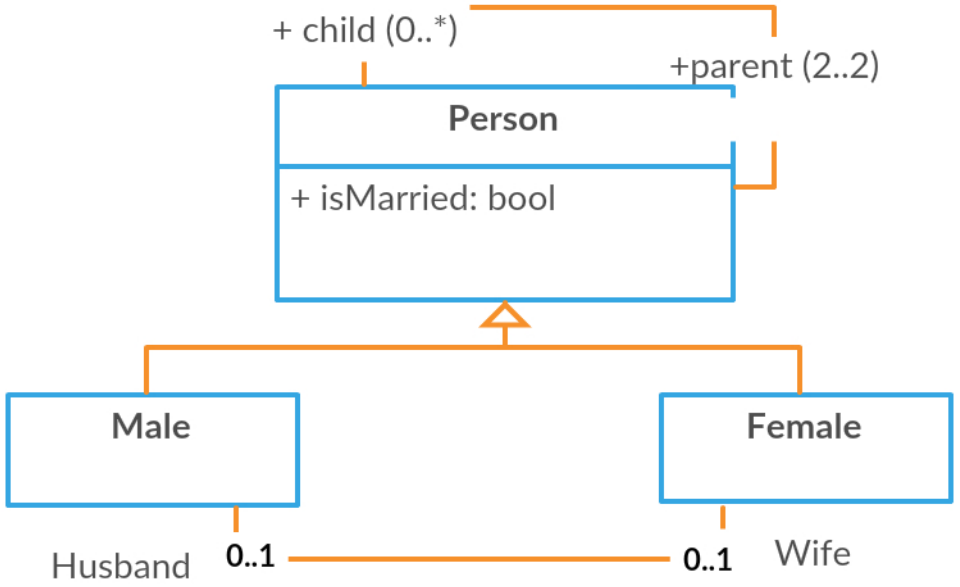
  } Concrete syntax

  - The meaning of the model structure

  } Semantics

HOW TO DESCRIBE?

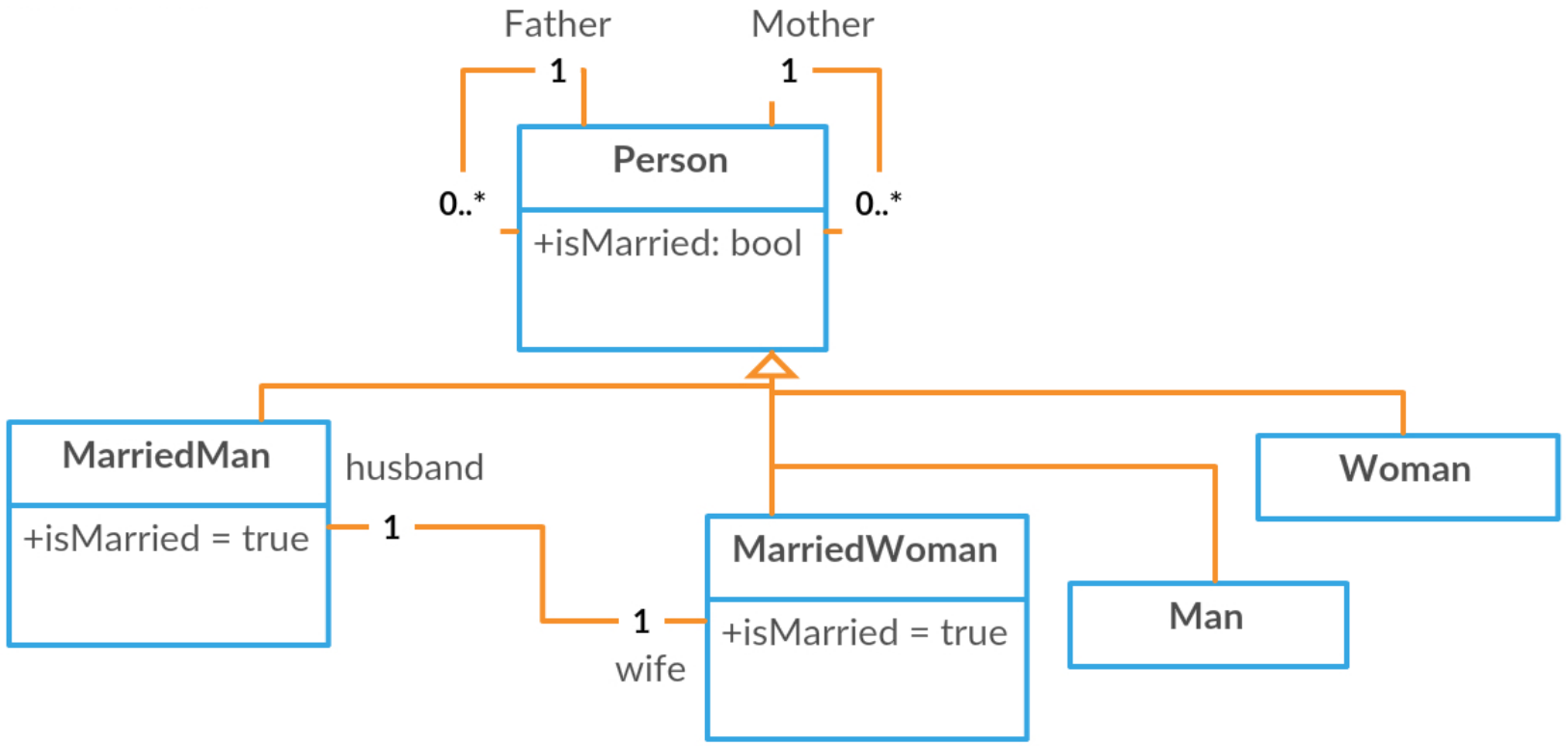




- A person has exactly one father and mother
- If a person has a wife, they are married

HOW TO DESCRIBE?

- Contains the previous two constraints, but not feasible



MOTIVATION

- Problem: description of complex constraints
  - > Two values depending on each other
    “A book contains exactly as many letters, as all its pages combined”
- Complex restrictions
  „A class may consist of boys and girls in any distribution, but no more than 35 students combined.”

CONTRAINTS

- What is a constraint?
  - *A constraints is a restriction on one or more elements or values of a metamodel.*
- Structural description is comfortable, but describing complex constraints in the structure is problematic
  - This shortcoming **does not** occur *because* the metamodeling language is wrongly constructed!
- How to specify constraints?

THANK YOU FOR YOUR ATTENTION!