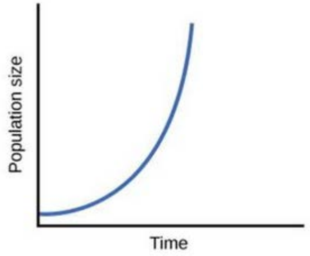
Software Engineering Group
QUERCUS
UNIVERSIDAD DE EXTREMADURA

An introduction to MDE: From toy to real-world projects in different application domains

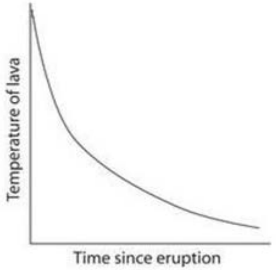
Cristina Vicente-Chicote
Quercus Software Engineering Group (QSEG)
Universidad de Extremadura
cristinav@unex.es

I-MDE-A Workshop - IMDEA Software (Madrid) - 16 May 2023

What comes to your mind when you hear the word **MODEL**?



$$y = a * e^{bx}$$

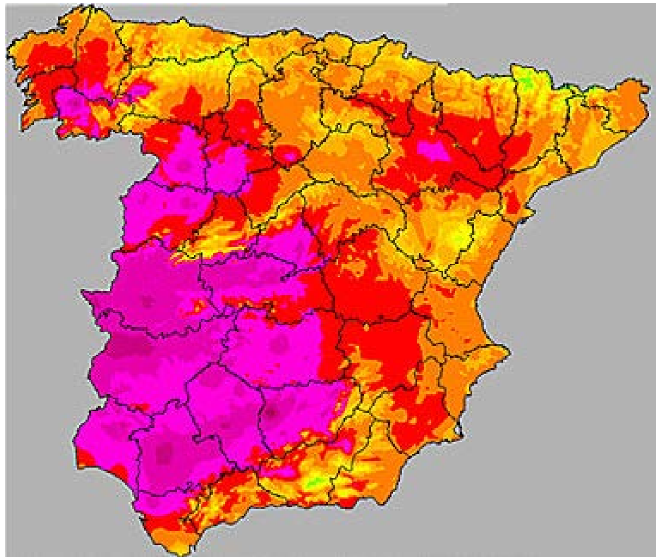
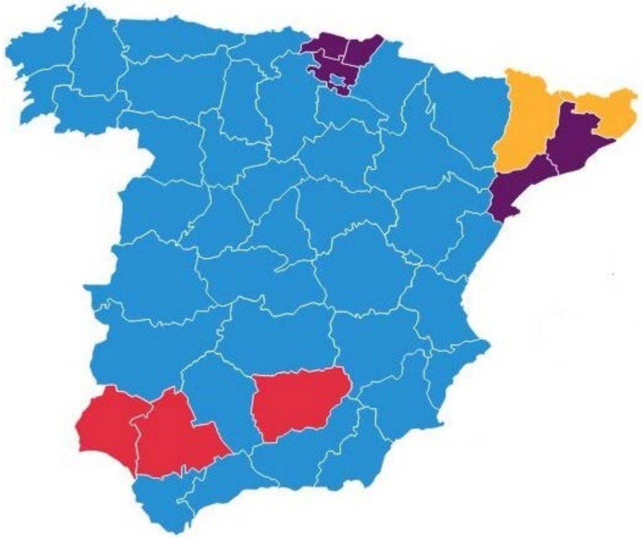


$$y = a * e^{-bx}$$

https://wordart.com/

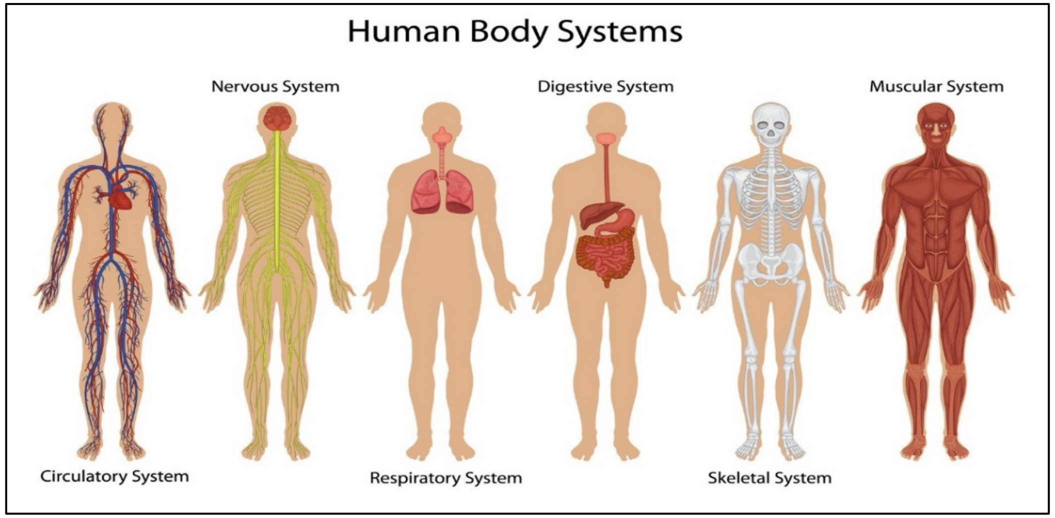
What is a model?

- ✓ A model is a **simplified representation** of a certain reality [Bezivin, 2005]
- ✓ We can build different models of the same reality with different **purposes**.

What is a model?

- ✓ A model is a **simplified representation** of a certain reality [Bezivin, 2005]
- ✓ We can also define alternative/complementary models according to different **viewpoints**, i.e., paying attention to certain features/parts. Each of these models will provide us with a partial/specific **view**

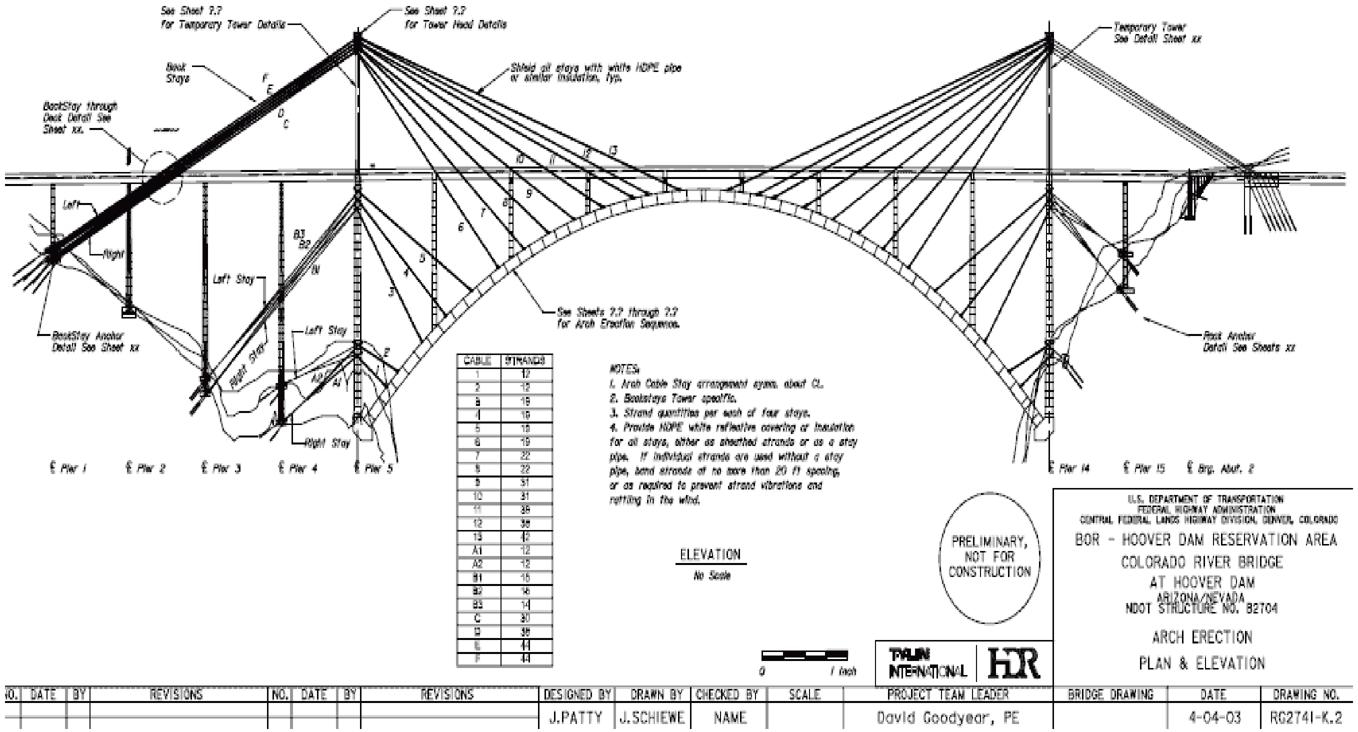


Views associated to the human body systems (T. Gander)
Source: https://goo.gl/images/bAc3Pa

What is a model?

- ✓ B. Selic identifies five key features for a model to be considered useful and effective [Selic, 2003]:
  - **Abstraction** → Represent a simplified/reduced version of the original system
  - **Understandability** → Easy to understand by the intended users
  - **Accuracy** → Offer a faithful representation of the original system
  - **Predictiveness** → Useful for reasoning about the original system
  - **Inexpensiveness** → It should be easier/cheaper/faster to develop than the original system

Models in Engineering



Models/diagrams/planes have been traditionally used in engineering for different purposes:

- ✓ To understand existing systems
- ✓ To specify, share and discuss with others the design of a new systems
- ✓ As a guide for system implementation
- ✓ As a prototype of a system to be built allowing us to detect errors, demonstrate or infer properties, etc. before implementing the actual system

Models in Software Engineering

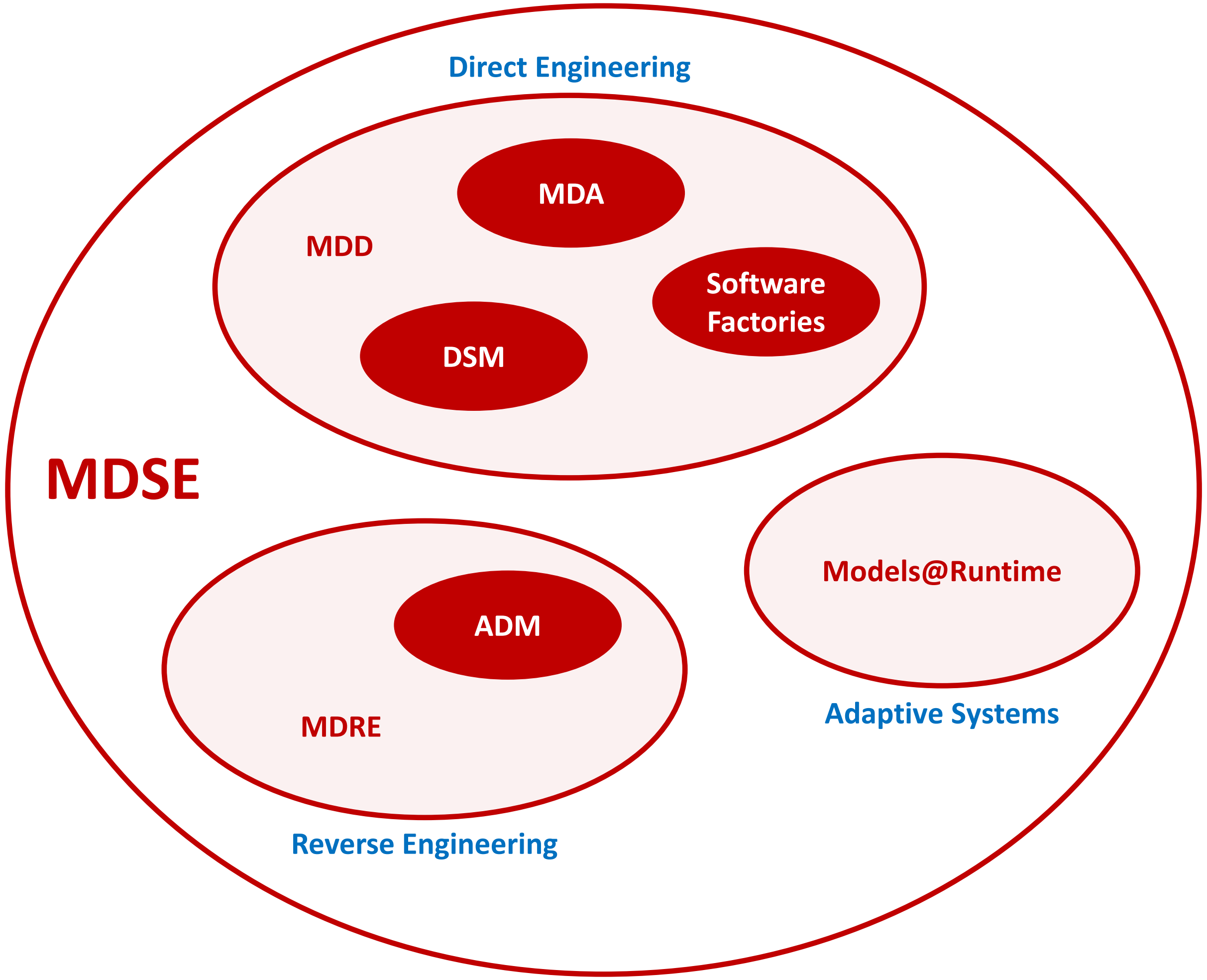
UML is probably the most widely known and spread in use software modeling language. In fact, it is claimed to be the *de facto* standard for software system modeling.

Limitations:

- ✓ UML models have been used (nearly exclusively) as documentation
- ✓ There is an important gap between models and actual system implementations due to…
  - The semantic gap between modeling and programming languages
  - The lack of tools supporting traceability and automated change propagation (model ↔ implementation)
- ✓ In most cases, models gathering different views of the system are not appropriately harmonized
- ✓ There is a lack of languages and tools enabling model management
  - Several model editors are available, but there is a lack of model compilers, code generators, model validators/simulators/optimizers, etc.

Model-Driven Software Engineering

Model-Driven Software Engineering (MDSE) is much more than just UML…



MDSE: Model-Driven Engineering

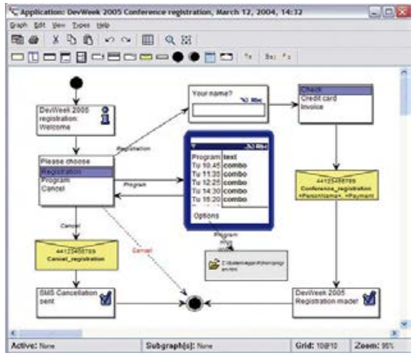
- ✓ **MDD**: Model-Driven Development (Direct Engineering)
  - **MDA**: Model-Driven Architecture (1)
  - **DSM**: Domain-Specific Modeling
  - Software Factories
- ✓ **MDRE**: Model-Driven Reverse Engineering (Reverse Engineering)
  - **ADM**: Architecture-Driven Modernization (2)
- ✓ Adaptive Systems
  - Models@Runtime

(1) http://www.omg.org/mda/

(2) http://adm.omg.org/

Model-Driven Software Engineering

- ✓ All MDSE approaches aim at…
  - Helping software developers to address the complexity of current software platforms and their increasing number of abstraction layers
  - Significantly reducing coding errors (compared to manual software implementation)
  - Increasing productivity in software development processes



Enabling the definition of new modelling languages

Automating code generation

Application code in a high-level programming language (C++, Java, etc.)
Application development framework (mobile, web, etc.)
Platform (OS, Hardware, etc.)

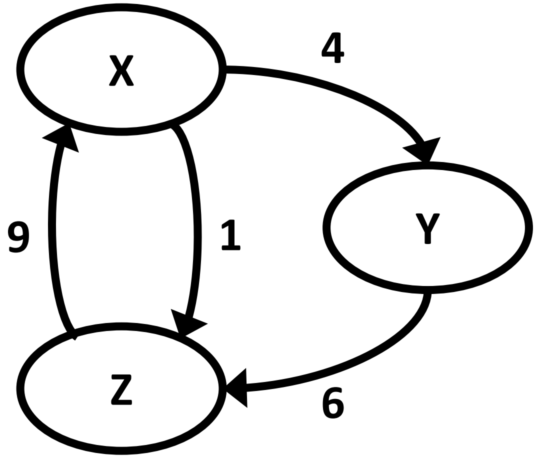
Model-Driven Software Engineering

- ✓ All the MDSE approaches share the following core features:
  - Each model represents (totally or in part) one aspect/view of a software system;
  - Each model is defined in terms of a modeling language, either a general-purpose language (e.g., UML) or a Domain-Specific Language (DSL);
  - A meta-model is used to formally define (the abstract syntax of) each modeling language;
  - Automation is typically achieved through the translation of models into code through model transformations.

Basic concepts

Model semantics

- ✓ Semantics (from the Greek term σημαντικό ς (semantikos) = “meaning”): Branch of linguistics concerned with meaning
- ✓ What does this model mean? What reality does it describe?

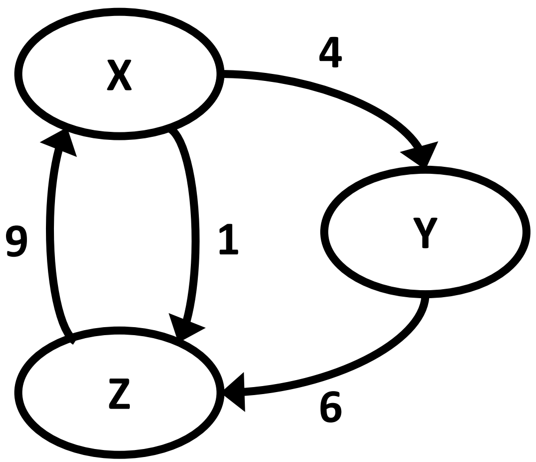


- Transitions among states after intervals of time (in secs)
- Migratory flows among countries (in millions of people)
- Payments among people (in Euros)
- …

Basic concepts

Model semantics → Interpretation

- ✓ The meaning of a model depends on its interpretation. For instance:
  - Ellipses may represent states/countries/people
  - Arrows may represent transitions/migratory flows/payments



One possible interpretation (meaning) of the previous model:

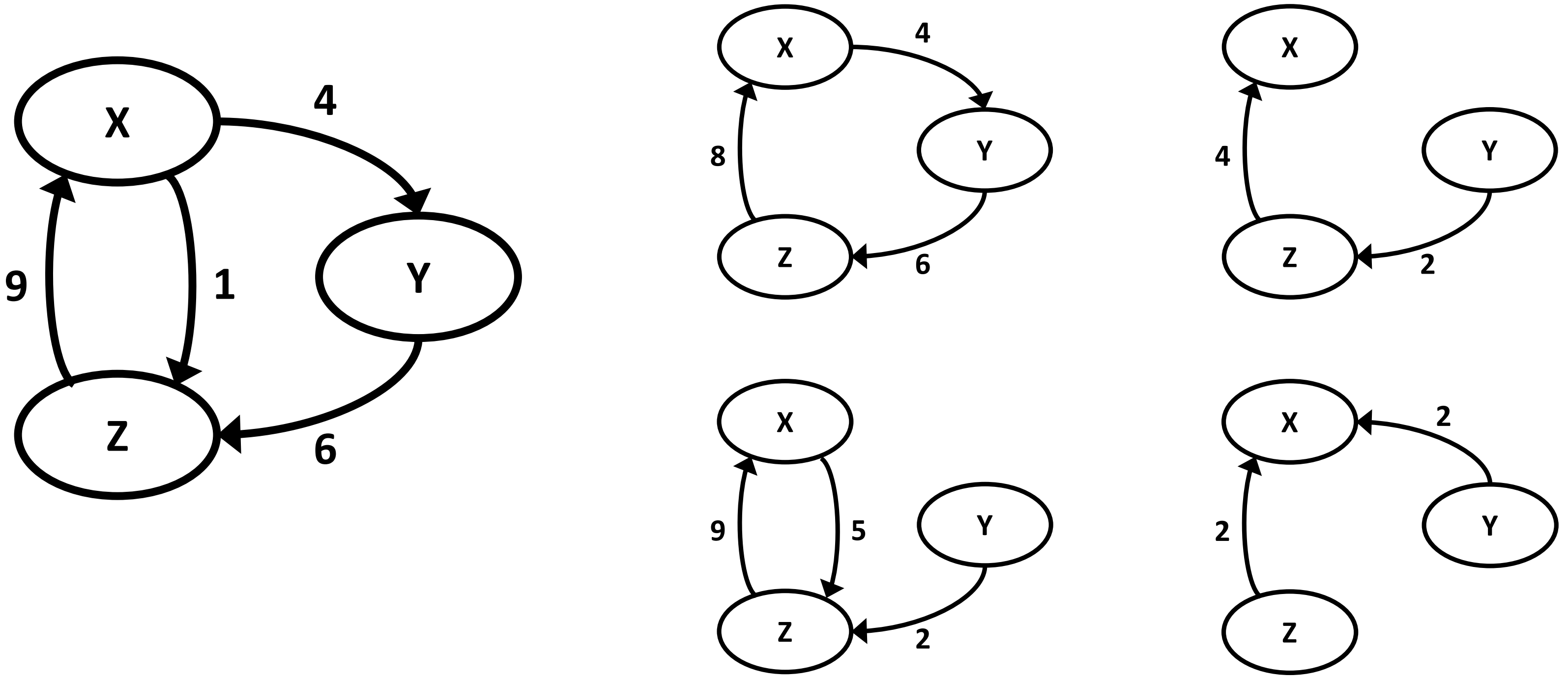
If X, Y and Z represent people and the arrows represent payments:

- X pays 4 € to Y and 1 € to Z
- Y pays 6 € to Z
- Z pays 9 € to X

Basic concepts

Model semantics → Transformation

- ✓ The meaning of a model also relates with model equivalence/derivation
- ✓ For instance, given the previous interpretation, all the models included next are equivalent and can be derived from the others:



Note that all these models share the following **invariant**:

- X receives 4 €
- Y pays 2 €
- Z pays 2 €

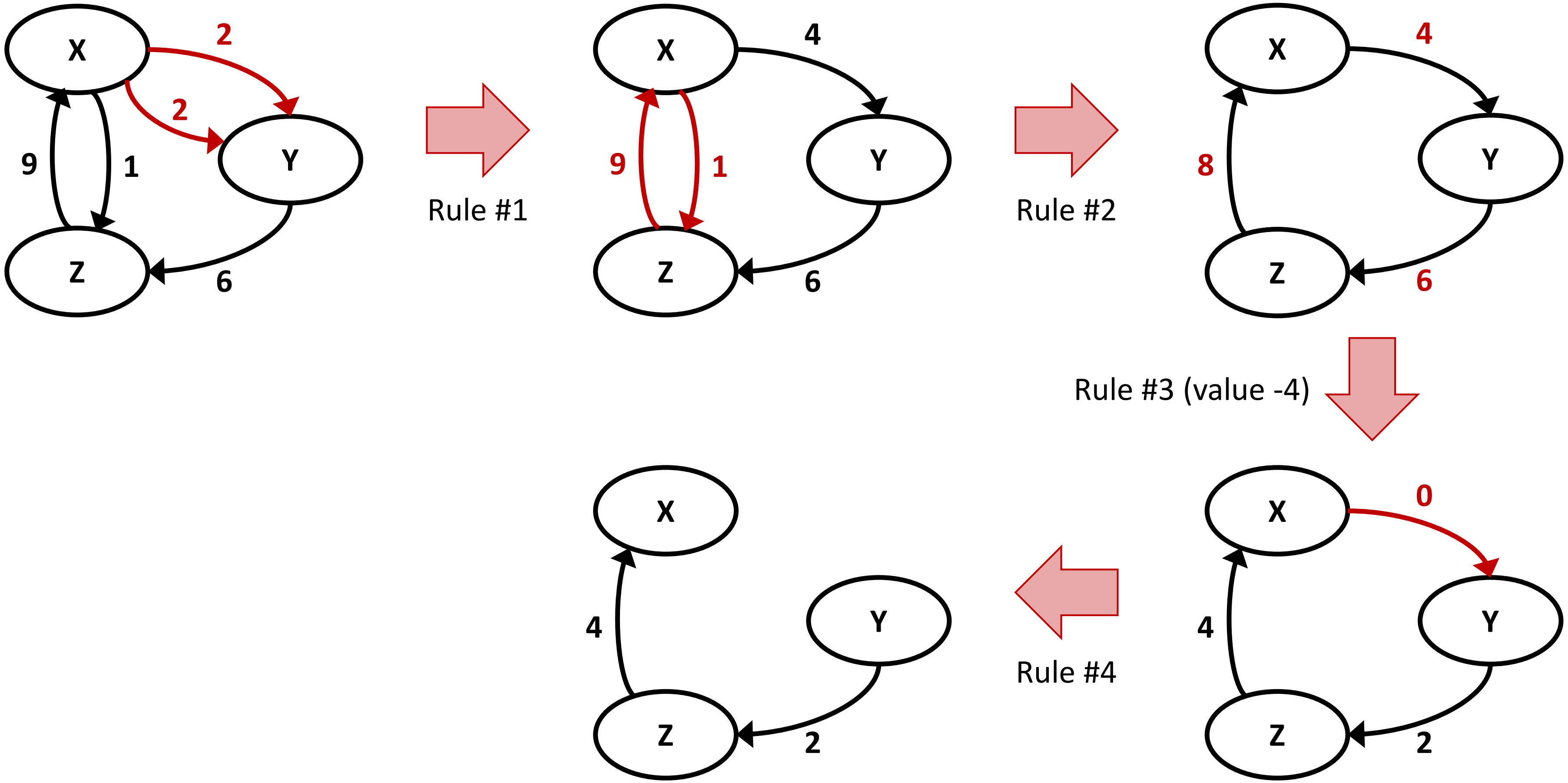
Basic concepts

Model semantics → Transformation

- ✓ "A **theory** is a way to deduce new statements about a system from the statements already included in a model of such system" [Seidewitz, 2003]
- ✓ A theory is a set of deductive transformation rules that allow us to derive models from other models
- ✓ Example: "The debt theory"
  - **Rule #1 (addition)**: two arrows *A1* (with value *v1)* and *A2* (with value *v2*), with the same source and target can be replaced by a single arrow with the same source and target as the original ones and with value *v1* + *v2*, and *vice versa*.
  - **Rule #2 (difference)**: two arrows *A1* (with value *v1)* and *A2* (with value *v2*), with opposite source and target can be replaced by a single arrow:
    - Alternative 1: with the same source and target as *A1* and value *v1* – *v2*
    - Alternative 2: with the same source and target as A2 and value *v2* – *v1*.
  - **Rule #3 (cycle)**: The value of the arrows being part of a cycle can be all increased (or decreased) with a constant value.
  - **Rule #4 (null arrow)**: Arrows with value = 0 can be removed / added between any source and target.

Basic concepts

Model semantics → Transformation



Basic concepts

Model semantics

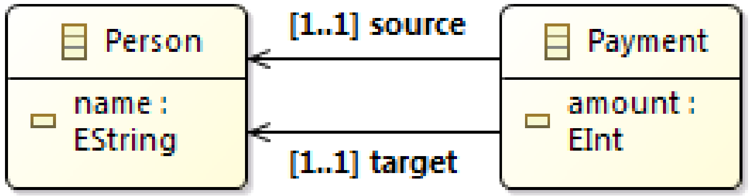
- ✓ Thus, in order to understand the meaning (semantics) of a model we must take into account:
  - How its concepts relate with the those being modelled (*interpretation*)
  - How it relates to other models (described using the same or a different representation) that can be obtained from it (*transformation*)
- ✓ ***Interpretation*** relates to the so-called ***denotational semantics***, while
- ✓ ***Transformation*** relates to the so-called ***operational semantics***

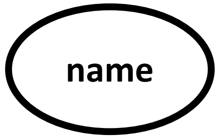
Basic concepts

Model syntax

- ✓ ***Syntax***: arrangement of words and phrases to create well-formed sentences in a language (Oxford)
- ✓ ***Abstract syntax***: Set of valid terms (***dictionary***) + set of rules that explain how to combine them to create correct sentences (***grammar***).
  - In the context of MDE, the abstract syntax of a modeling language is usually defined using a ***meta-model***. Alternative representations may be found, e.g., based on BNF/EBNF
- ✓ ***Concrete syntax*** (a.k.a., ***notation***): Set of (graphical or textual) symbols used to represent the modeling concepts defined in the abstract syntax.
  - Each modeling language has a unique abstract syntax, but there might be more than one concrete syntax built on it

Abstract syntax (meta-model)

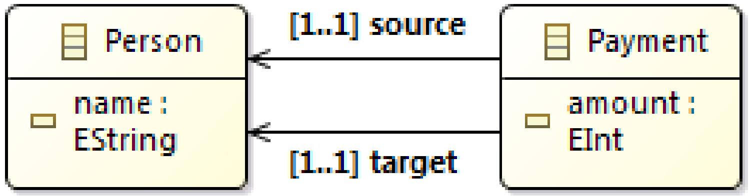


Terms from the abstract syntax	Concrete graphical syntax 1	Concrete graphical syntax 2
Person	 **name**	 **name**
Payment	**amount** 	***amount €*** 

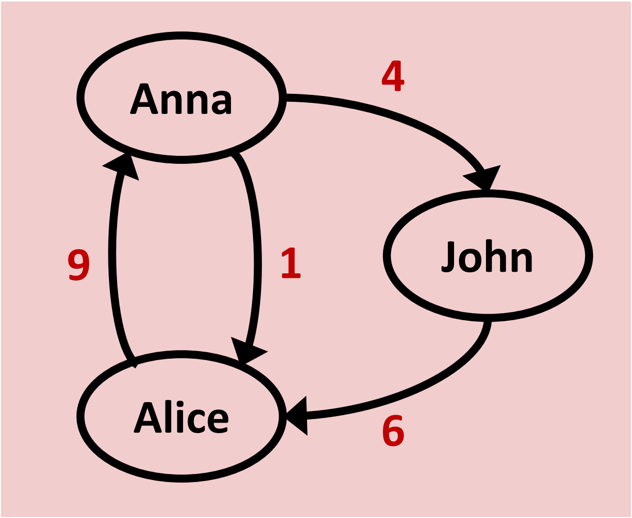
Basic concepts

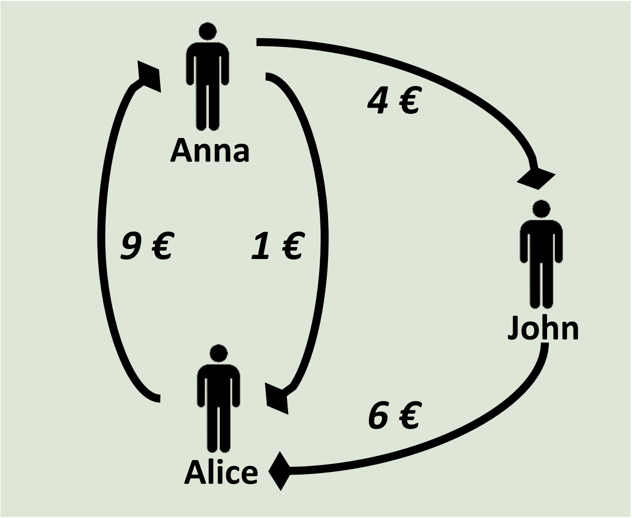
Model syntax

Abstract syntax (Meta-Model)



Terms from the abstract syntax	Concrete graphical syntax 1	Concrete graphical syntax 2
Person	**name**	**name**
Payment	**amount**	***amount €***

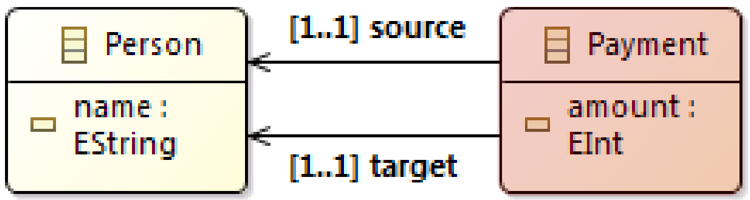




Basic concepts

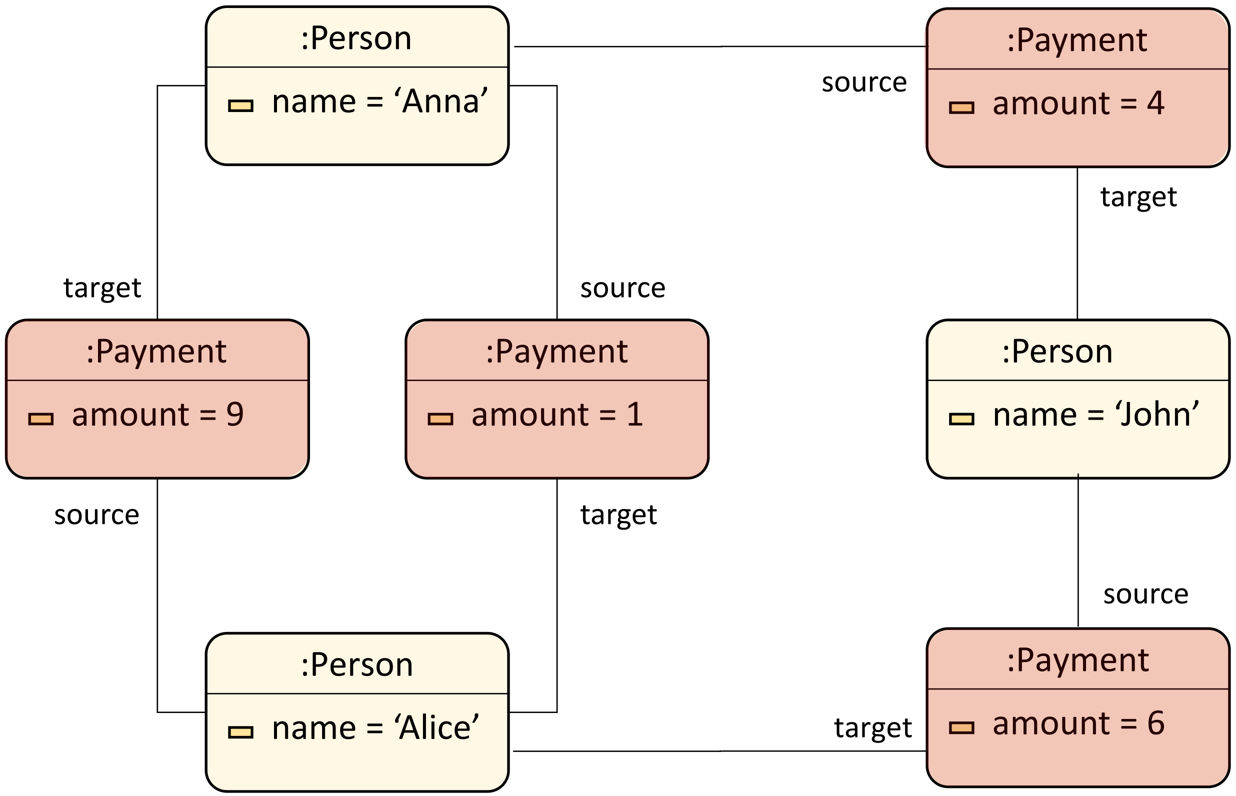
Model syntax

Meta-Model



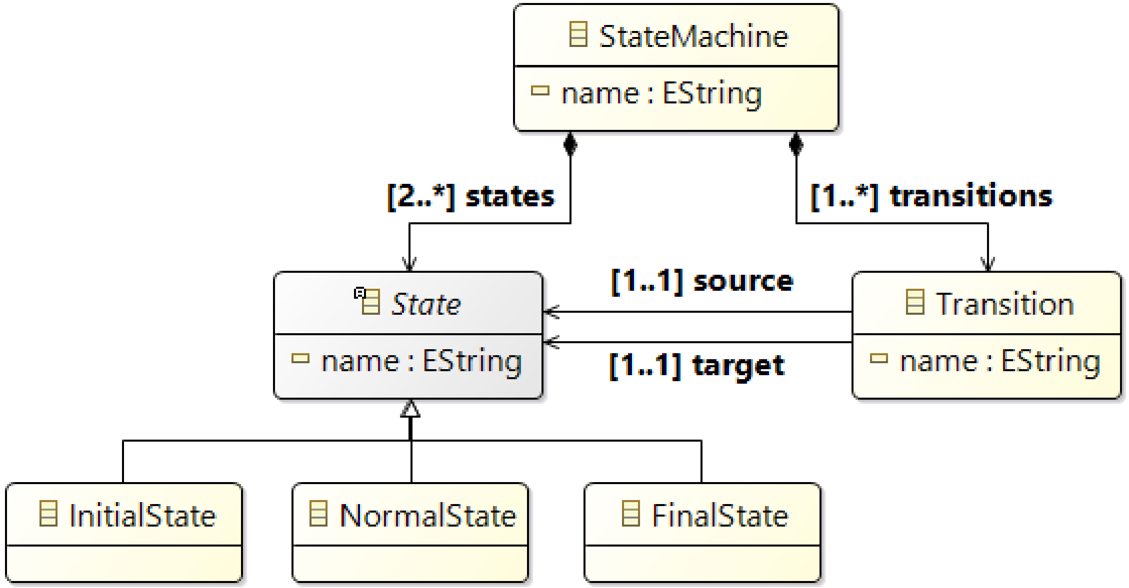
Model

Defined in terms (as an instance) of the meta-model



Basic concepts

Meta-modelling



- ✓ **Meta-classes**: StateMachine, State (abstract), Transition, InitialState, NormalState, FinalState
- ✓ **Attributes**: StateMachine.*name*, State.*name*, Transition.*name*
- ✓ **Compositions**: StateMachines **contain** *states* and *transitions*
- ✓ **References**: Each Transition **has** a *source* (State) and a *target* (State)
- ✓ **Generalization**: InitialState, NormalState and FinalState **are** States

Basic concepts

Additional language constraints

- ✓ Most times, UML-like class diagrams are not expressive enough to define all the relevant aspects of a modelling language.
- ✓ Frequently, it is necessary to define additional constraints (a.k.a. invariants) to be hold by the systems being modeled (*well-formedness rules*).
- ✓ These constraints are usually specified using OCL (Object Constraint Language)
- ✓ Back to the State Machine example, how can we avoid reflexive transitions (i.e., from a state to itself)?

```
context Transition
    inv:   ReflectiveTransitionsNotAllowed
           self.source <> self.target
```

Basic concepts

Syntax + Semantics

- ✓ **Modeling language**
  - **Semantics**
    - **Interpretation** (semantic correspondence)
      Defines the meaning of the language elements in terms of real-world concepts
    - **Transformation** (deductive theory)
      Relates equivalent models via deductive/transformation rules
  - **Syntax**
    - **Abstract**: logical structure of correct models (terms + grammatical rules)
    - **Concrete**: textual or graphical notation
- ✓ The **concrete syntax** depends on the **abstract syntax**
- ✓ Syntax and semantics are closely related. The syntax determines which expressions are correct, while the semantics provides non-ambiguous meaning to those expressions. The semantics of a language is not embedded in its syntax (i.e., in its meta-model) [Harel, 2004]

Basic concepts

Domain Specific Languages (DSL)

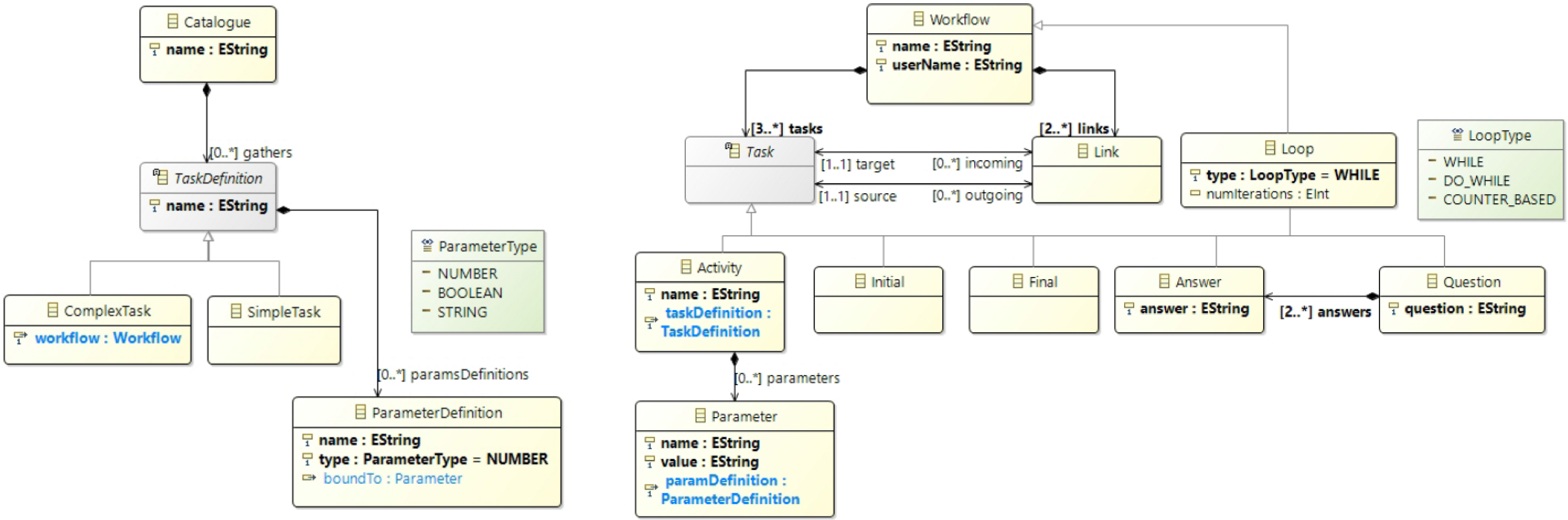
- ✓ A **Domain-Specific Language** (DSL) is a modeling language, either textual or graphical, used to describe a particular semantic domain, e.g., a particular application domain
- ✓ All modeling languages are somehow domain-specific, although they may cover wider or narrower domains. For instance, UML is claimed to be a general-purpose (rather than a domain-specific) modeling language. However, it is somehow restricted, not to a particular application domain, but to object-oriented software development approaches.
- ✓ The abstract syntax of a DSL gathers the concepts relevant for modeling the target domain. These concepts must have a clear correspondence with those in the semantic domain (i.e., concepts with a clear meaning for the domain experts using the DSL). Thus, it is essential to select appropriate and unambiguous terms (and their corresponding graphical/textual representation) when defining the syntax of a DSL.

“Toy” MDE projects

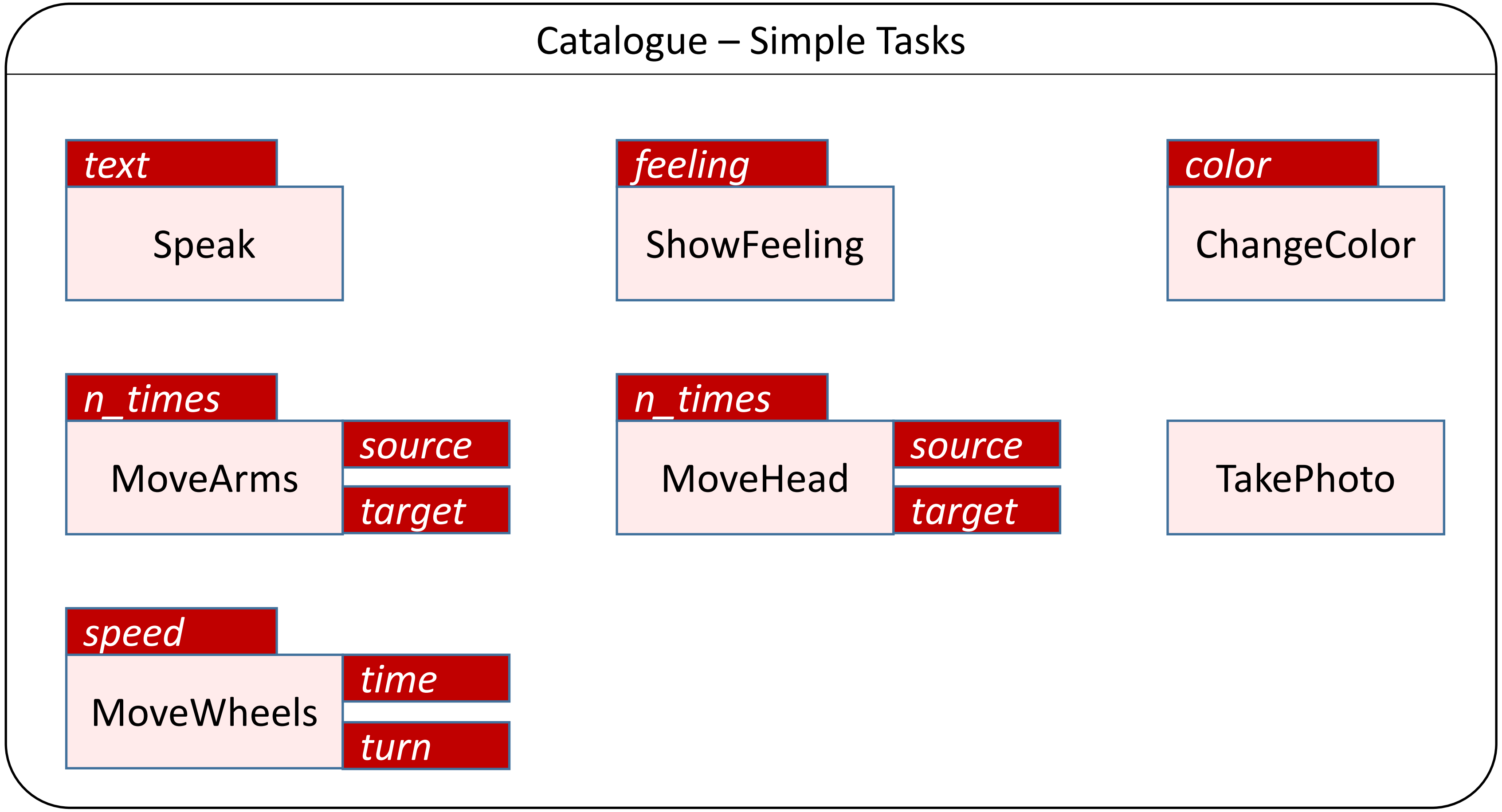
PiLHaR: A tool aimed to ease the definition, composition and execution of edutational robot workflows

- ✓ **Project goal**: provide a graphical editor allowing therapists working with autistic children to easily define task workflows to be executed in an educational robot.
- ✓ Bachelor student: **Gloria Díaz-González**
- ✓ Supervisors: Cristina Vicente-Chicote, José Ramón Lozano-Pinilla.
- ✓ Material available at: https://github.com/GloriaDG22/GeneracionCodigoCozm

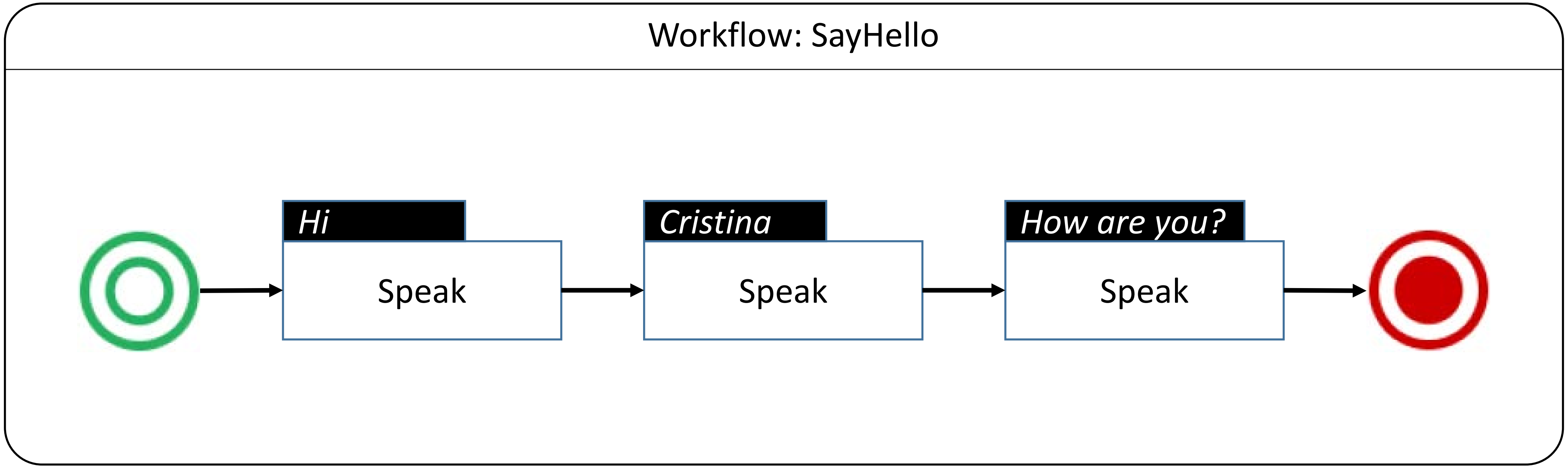
PiLHaR: A tool aimed to ease the definition, composition and execution of edutational robot workflows



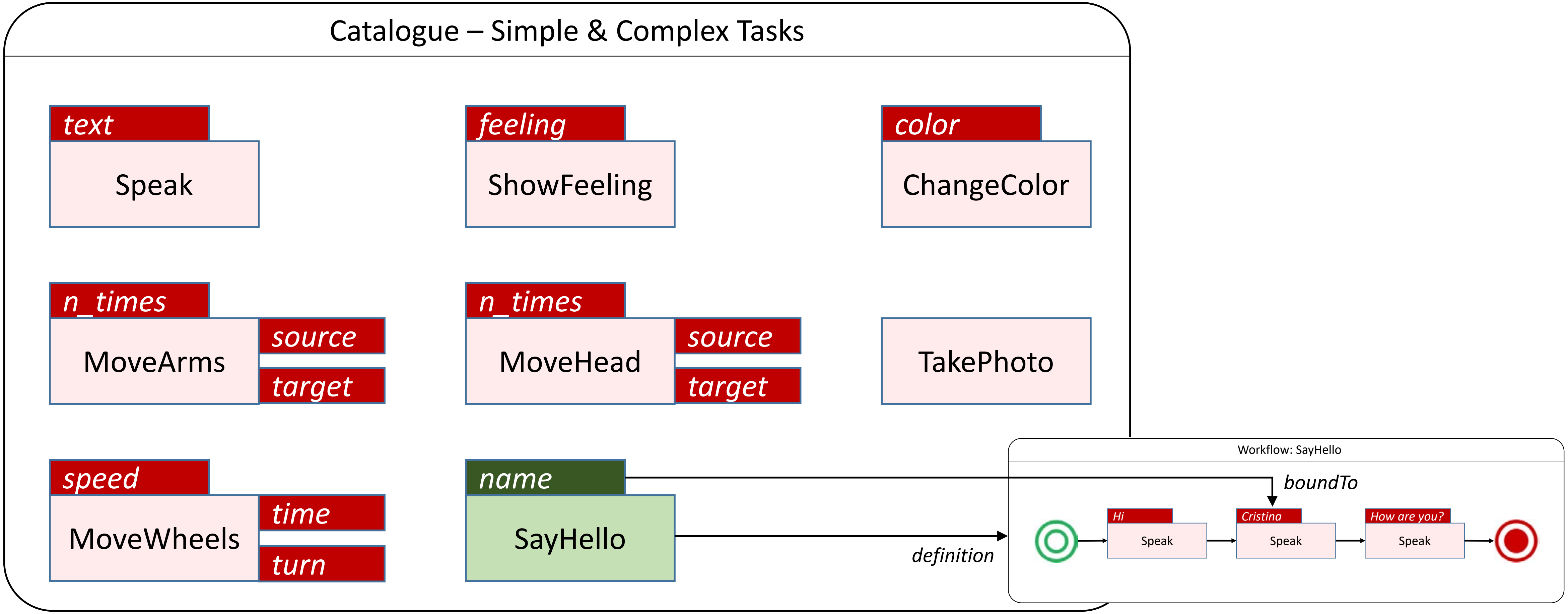
PiLHaR: A tool aimed to ease the definition, composition and execution of edutational robot workflows



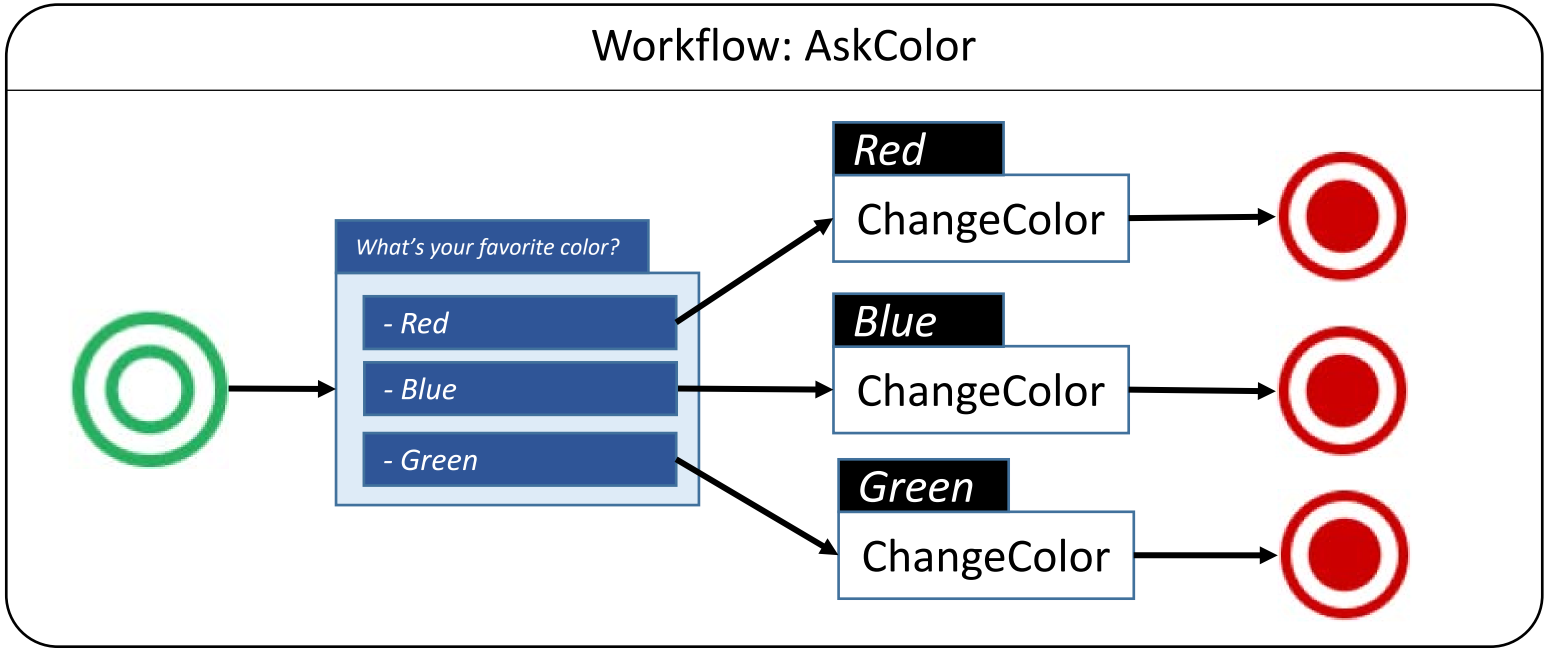
PiLHaR: A tool aimed to ease the definition, composition and execution of edutational robot workflows



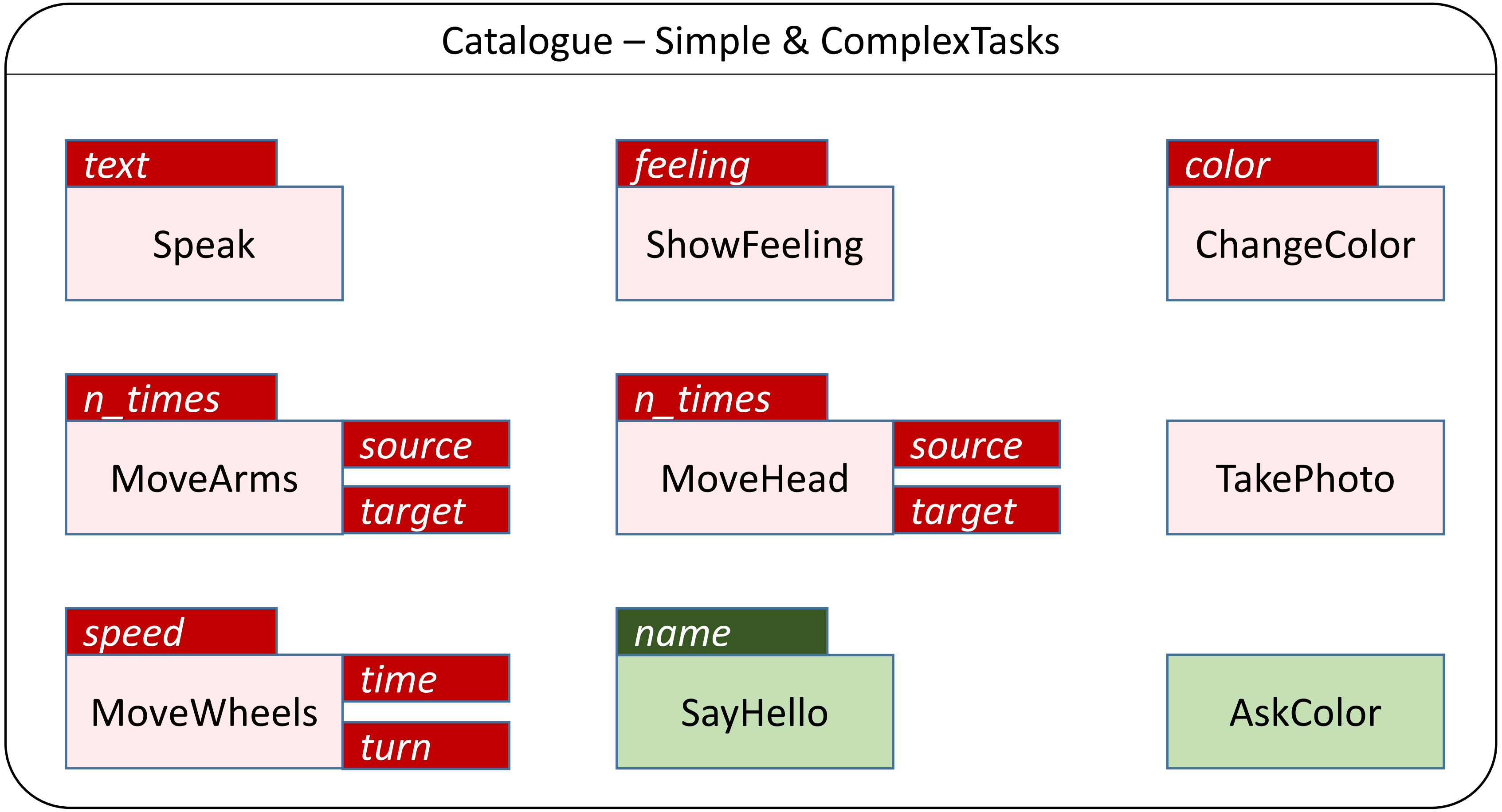
PiLHaR: A tool aimed to ease the definition, composition and execution of edutational robot workflows



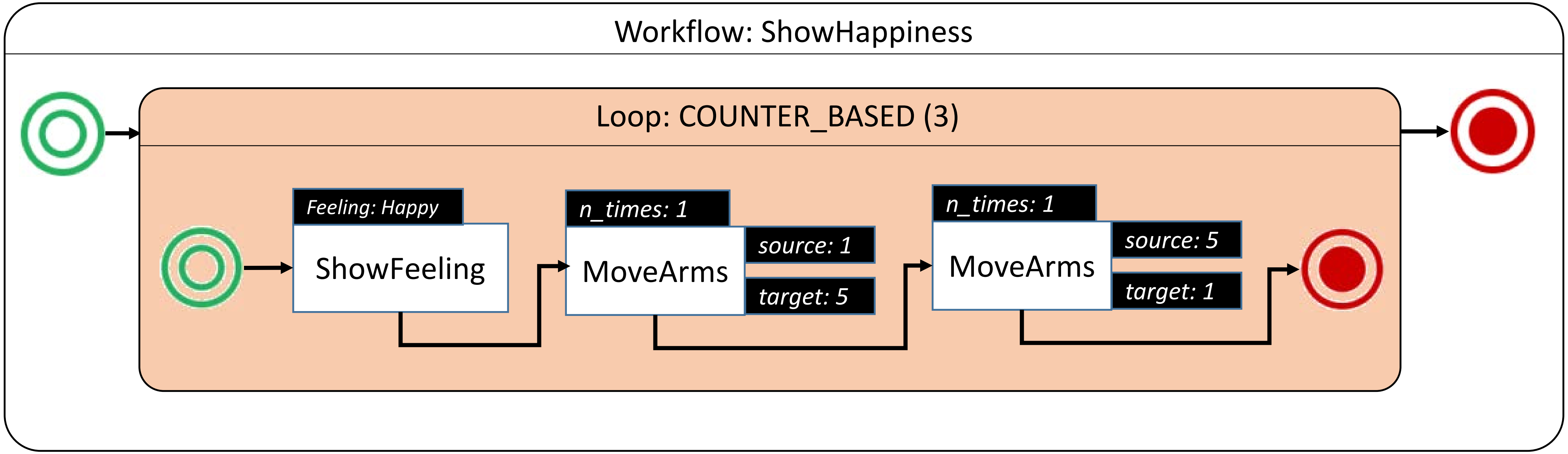
PiLHaR: A tool aimed to ease the definition, composition and execution of edutational robot workflows



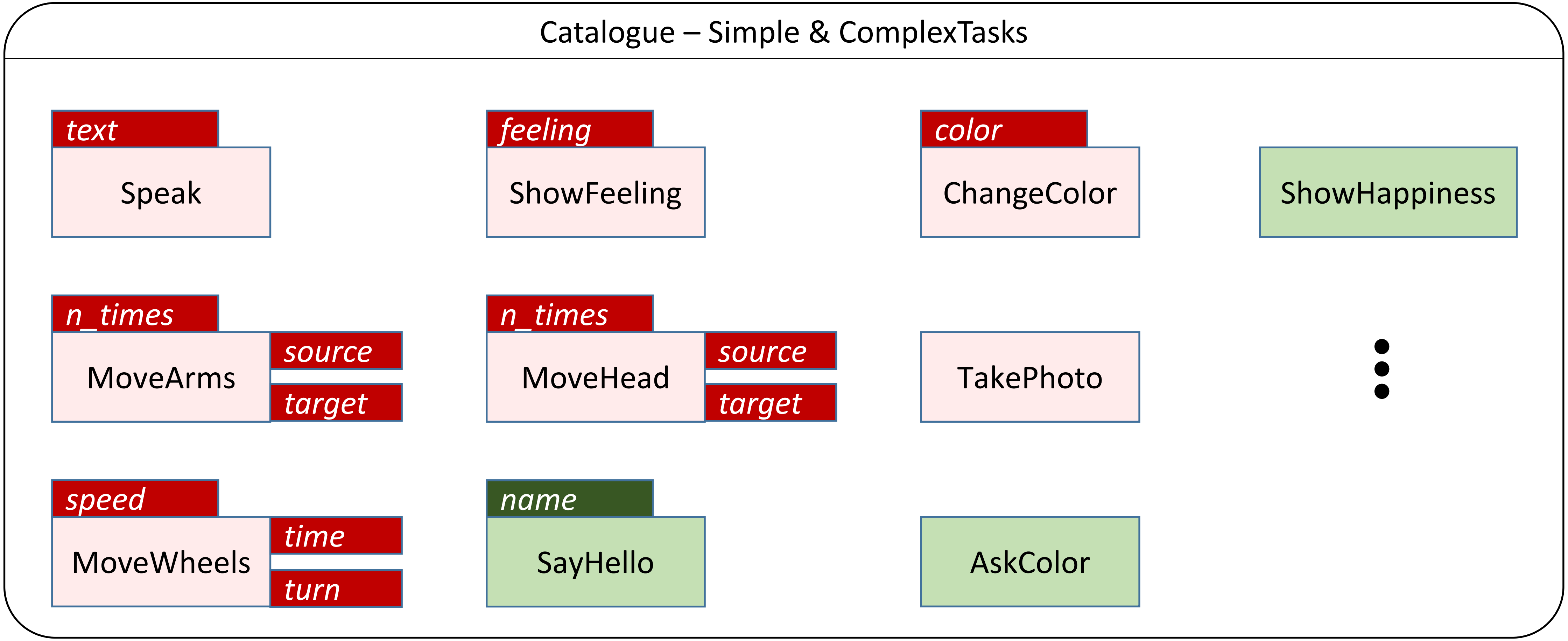
PiLHaR: A tool aimed to ease the definition, composition and execution of edutational robot workflows



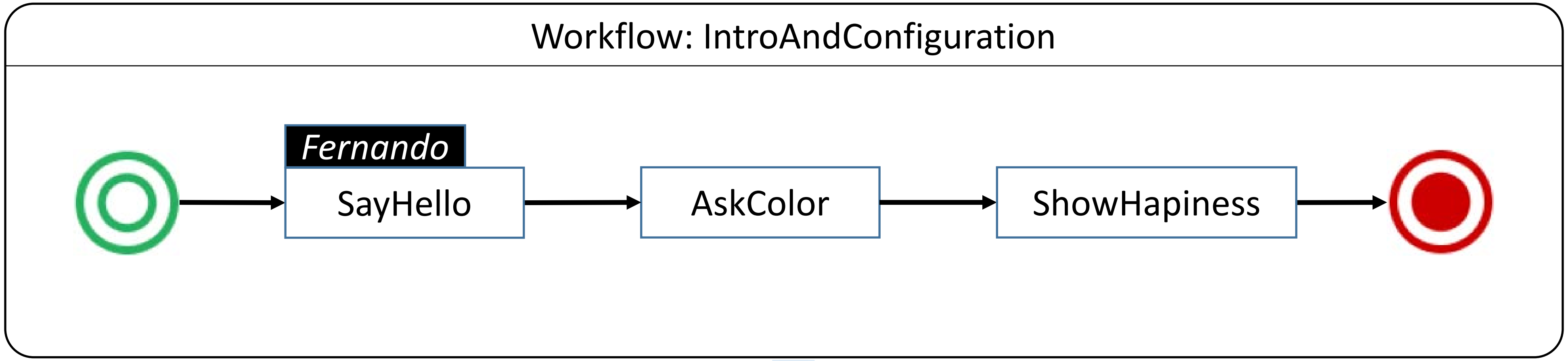
PiLHaR: A tool aimed to ease the definition, composition and execution of edutational robot workflows

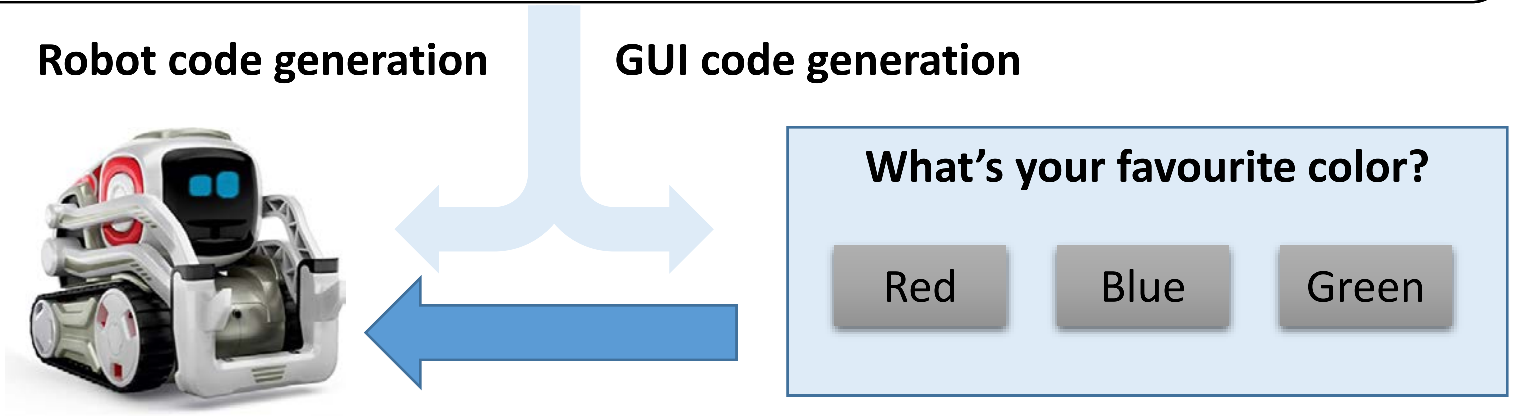


PiLHaR: A tool aimed to ease the definition, composition and execution of edutational robot workflows



PiLHaR: A tool aimed to ease the definition, composition and execution of edutational robot workflows





PiLHaR: A tool aimed to ease the definition, composition and execution of edutational robot workflows

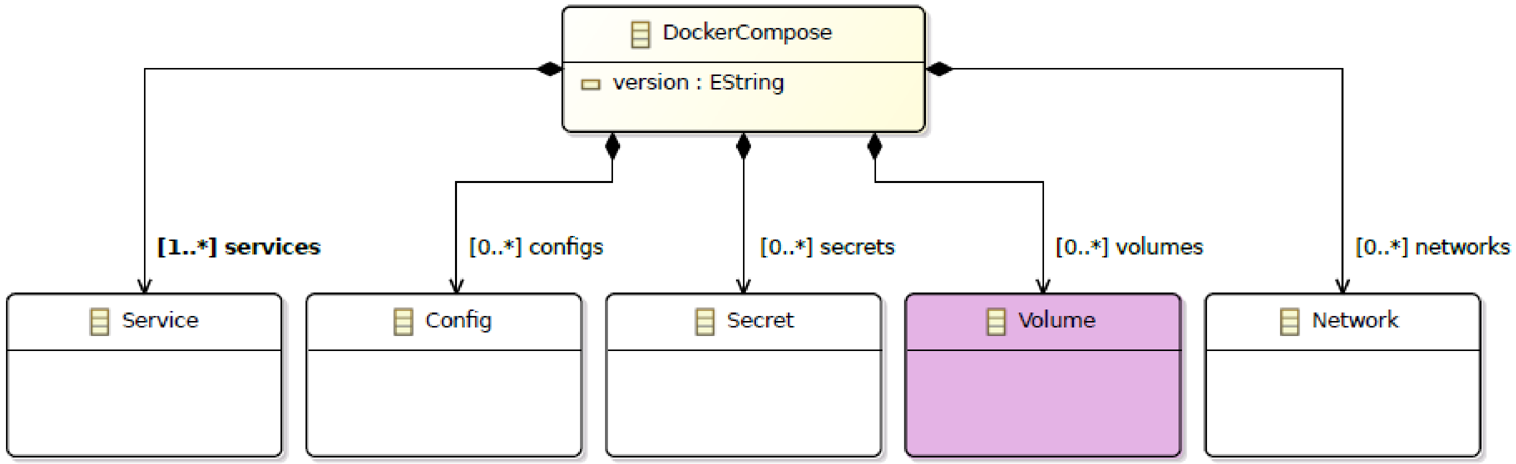
- ✓ **Results**
  - The therapists we worked with really appreciated the tool as it allowed them to incorporate *Cozmo* as part of their therapies. They found the possibility of reusing/configuring their workflows in different therapy routines with different children particularly useful.
  - Furthermore, they discovered that some of their children also loved programming the robot using PiLHaR ☺.
  - The project is been supported by a regional business acceleration program and it has been recently awarded with the “UEX excellence and social engagement” price.

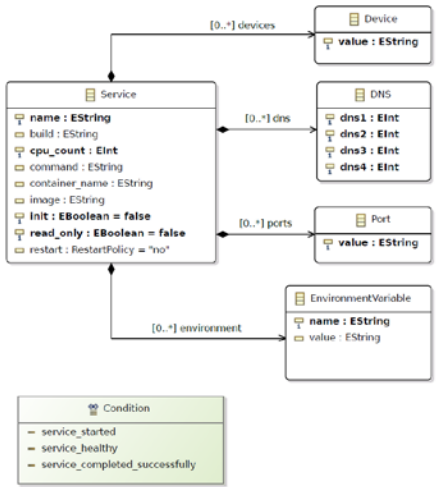
CML: modelling, validation and generation of container-based applications

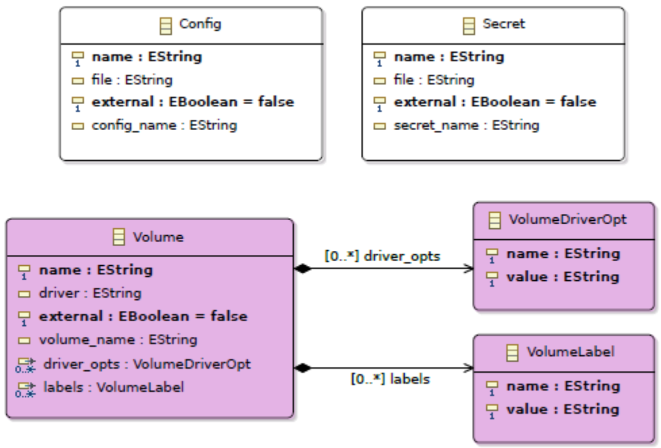
- ✓ **Project goal:** Provide software developers with a set of tools aimed at easing the specification, validation and visualization of Docker/Docker-Compose-based architectures.
  - Reduce the learning curve for novel developers.
  - Provide more experienced ones with new features, currently not supported by existing tools: automatic validation of the specifications, dual and synchronized graphic-textual representation, etc.
- ✓ Bachelor student: **Lorenzo G. Ceballos-Bru**
- ✓ Supervisors: Cristina Vicente-Chicote, José Ramón Lozano-Pinilla.
- ✓ Material available at: https://github.com/elpiter15/CML

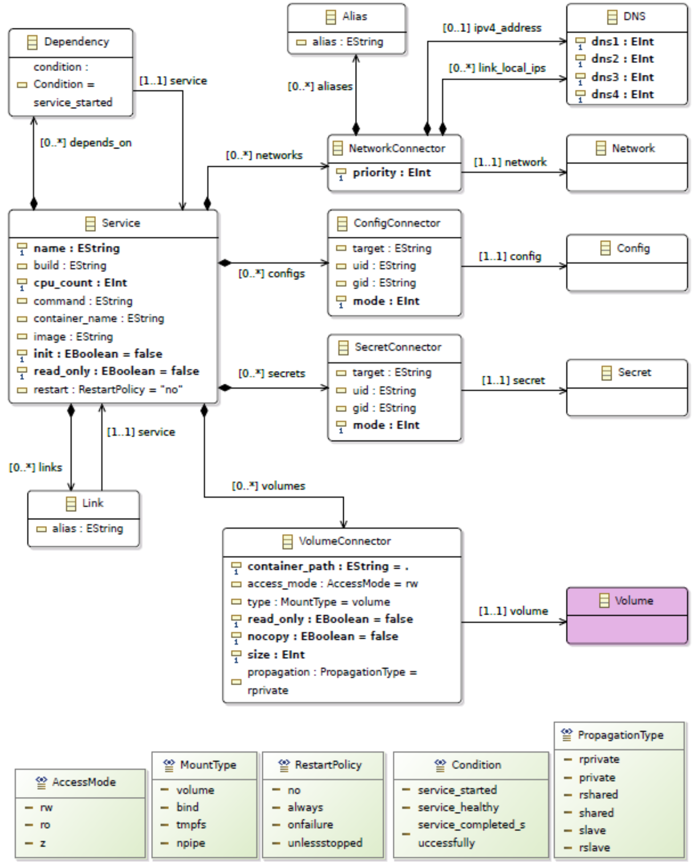
Software Engineering Group
QUERCUS
UNIVERSIDAD DE EXTREMADURA

CML: modelling, validation and generation of container-based applications





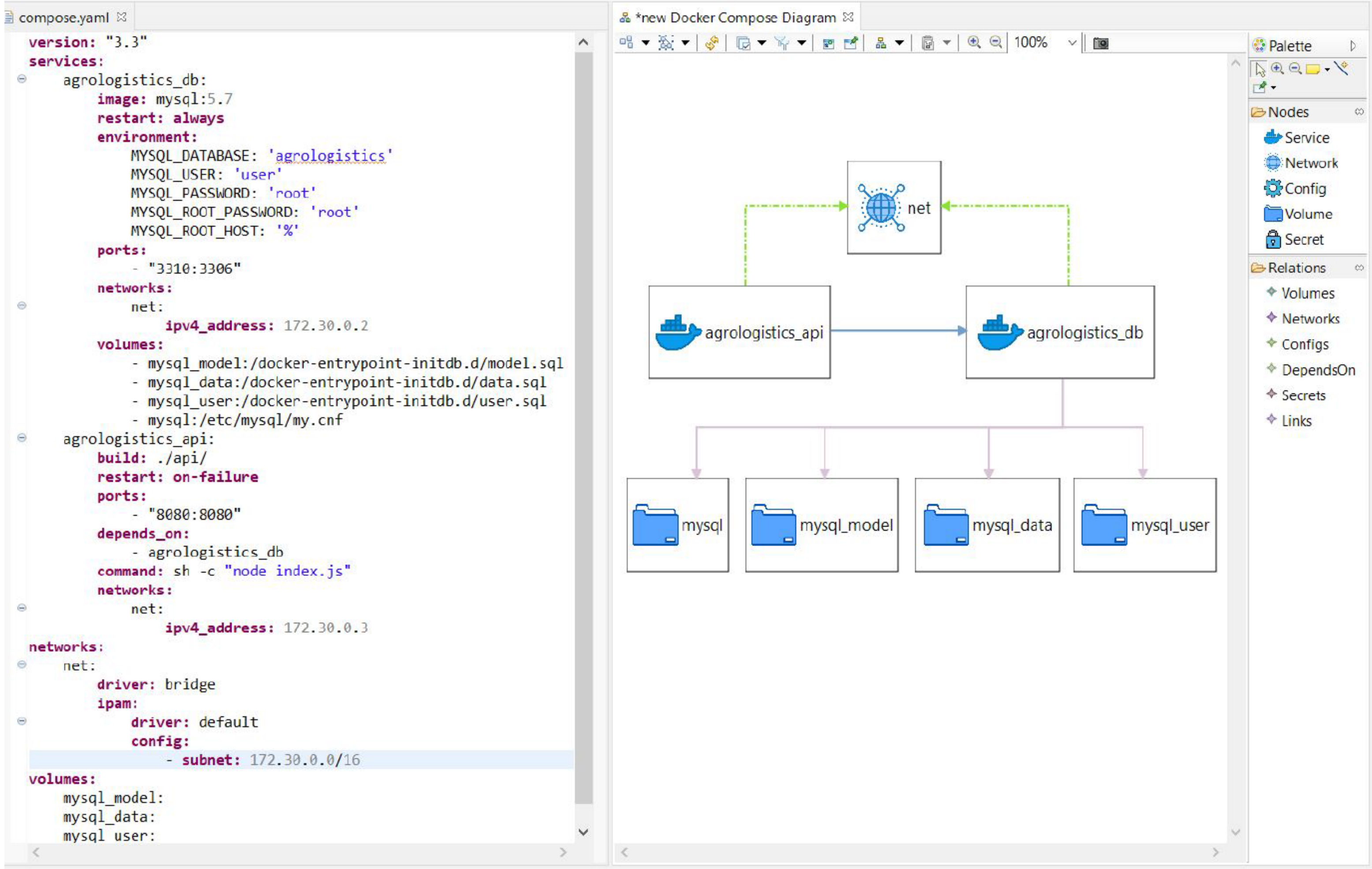




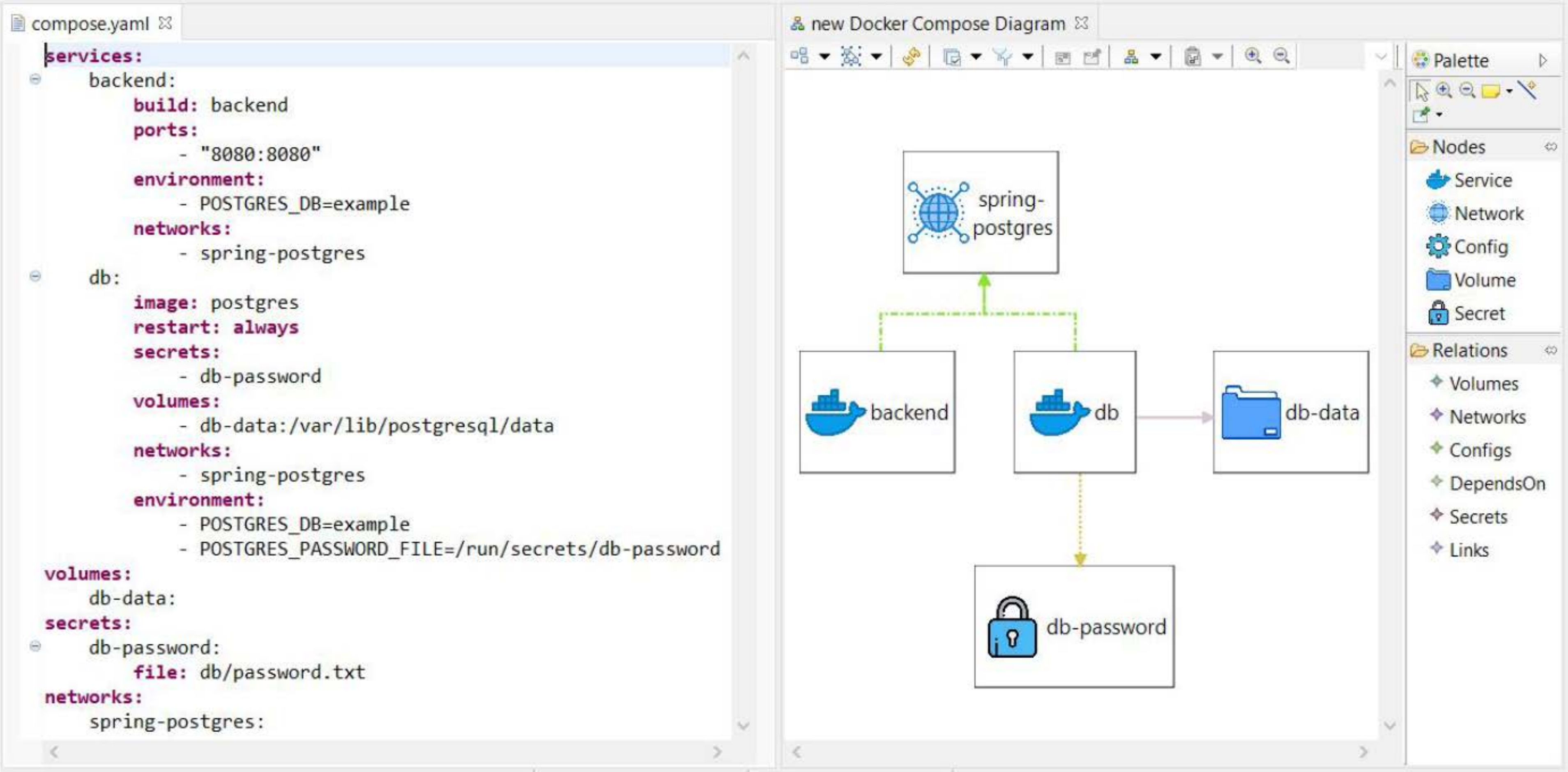
CML: modelling, validation and generation of container-based applications

```
grammar org.xtext.example.dockercompose.DockerCompose with org.eclipse.xtext.common.Terminals
import "http://www.eclipse.org/modeling/example/dockercompose/DockerCompose"
import "http://www.eclipse.org/emf/2002/Ecore" as ecore

DockerCompose returns DockerCompose:
(           ('version:' version=Version)?
        &  ('services:' (services+=Service)+ )
        &  ('volumes:' (volumes+=Volume)+ )?
        &  ('configs:' (configs+=Config)+ )?
        &  ('secrets:' (secrets+=Secret)+ )?
        &  ('networks:' (networks+=Network)+ )?
);

Service returns Service:
{Service}
        name=ID ':'
        (
            ('build:' build=PATH)?
          &  ('image:' image=Image)?
          &  ('cpu_count:' cpu_count=EInt)?
          &  ('command:' command=Command)?
          &  ('container_name:' container_name=EString)?
          &  ('restart:' restart=RestartPolicy)?
          …
```

CML: modelling, validation and generation of container-based applications



CML: modelling, validation and generation of container-based applications



Real-world MDE projects

RoQME: Dealing with non-functional properties through global Robot Quality-of- Service Metrics (H2020 RobMoSys Project)

- ✓ **Project goal**: provide software developers with (1) a modeling framework for specifying QoS metrics defined on non-functional properties (e.g., safety, performance, resource consumption, user engagement, etc.); and (2) a runtime infrastructure allowing them to estimate these metrics according with the perceived situation.
- ✓ **Project consortium**: UEX, UMA, Biometric Box
- ✓ General overview:
  - https://robmosys.eu/roqme/
  - https://robmosys.eu/wiki-sn-03/baseline:environment_tools:roqme-plugins
- ✓ Demo in an intralogistics scenario: https://robmosys.eu/wiki/community:roqme-intralog-scenario:start
- ✓ Project resources available at: https://github.com/roqme/robmosys-roqme-itp

RoQME: Dealing with non-functional properties through global Robot Quality-of- Service Metrics (H2020 RobMoSys Project)

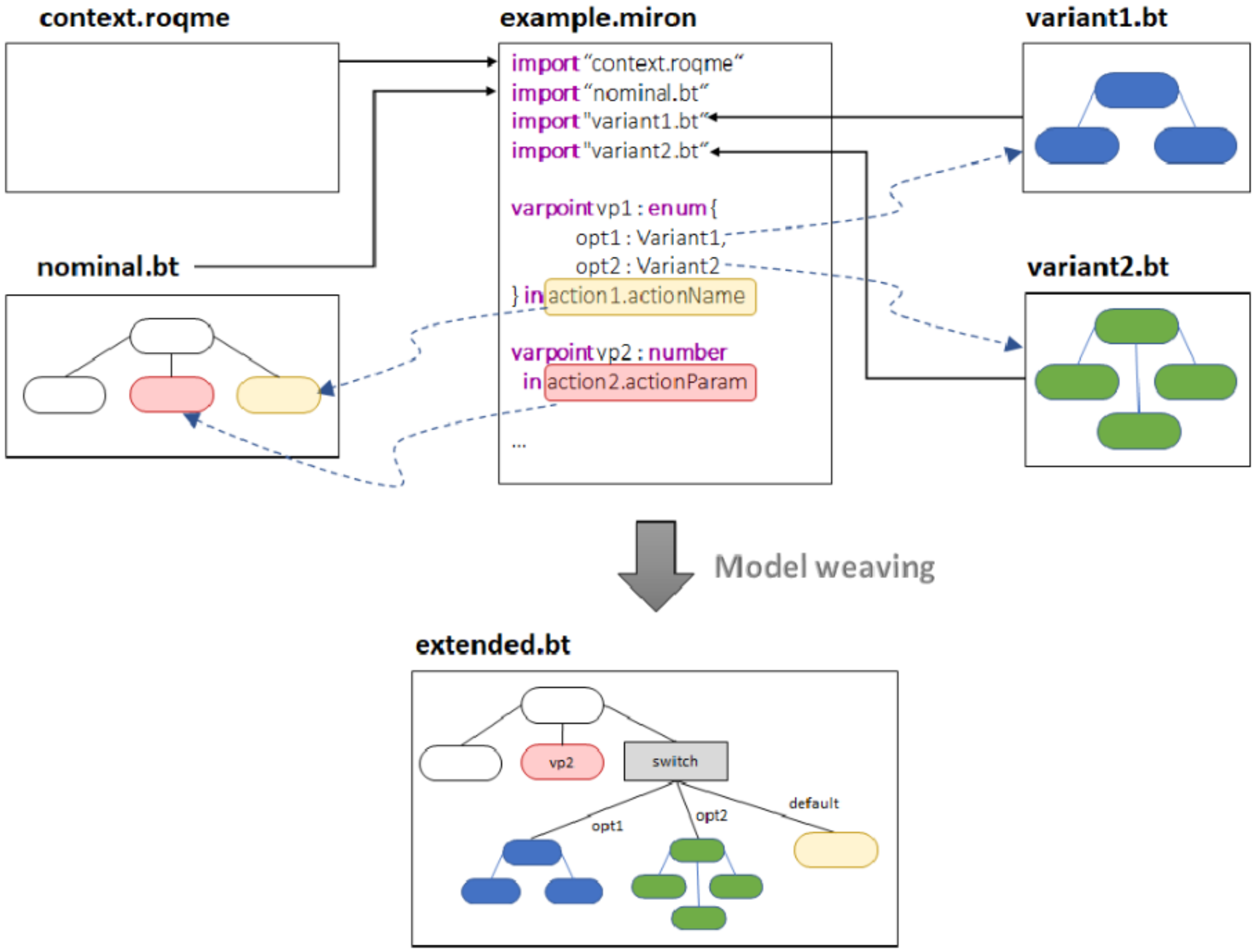
```
property Safety reference 1
property Performance reference 0.5

context Bump : eventtype
context Velocity : number
context PersonState : boolean
context JobState : enum {NOT_STARTED, STARTED, COMPLETED, ABORTED}
context RobotState : enum {IDLE, CHARGING, DRIVING_WITH_LOAD, DRIVING_EMPTY, ERROR }
context TimeJobDone : time := period (JobState::STARTED -> JobState::COMPLETED)

observation O1 : Bump undermines Safety VERY_HIGH
observation O2 : Velocity > MAX_V & PersonState  undermines Safety VERY_HIGH
observation O3 : JobState::COMPLETED while(TimeJobDone<AVG_JOB) reinforces Performance HIGH
observation O4 : RobotState::ERROR undermines Performance
observation O5 : JobState::ABORTED undermines Performance
```

RoQME: Dealing with non-functional properties through global Robot Quality-of- Service Metrics (H2020 RobMoSys Project)

MIRoN: QoS Metrics-In-the-loop for better Robot Navigation (H2020 RobMoSys Project)

- ✓ **Project goal**: provide a modeling framework allowing designers to endow robots with the ability of self-adapting their behaviour according to the situation perceived at runtime. MIRoN allows designers to model:
  - Behaviour Trees (BT), describing both nominal and alternative robot behaviours;
  - Variation points (linked to tasks/parameters in the BT models), which determine the decision space of the adaptation process;
  - Contexts, expressed in terms of RoQME QoS metrics; and
  - Adaptation policies, explicating how to configure the variation points (i.e., the robot behaviour) depending on the perceived situation (based on RoQME QoS metrics) in order to optimize relevant non-functional properties, such as safety or performance.
- ✓ **Project consortium**: UEX, UMA, Blue Ocean Robotics
- ✓ General overview: https://robmosys.eu/miron/
- ✓ Project resources available at: https://github.com/MiRON-project/Miron-Framework

MIRoN: QoS Metrics-In-the-loop for better Robot Navigation
(H2020 RobMoSys Project)



MIRoN: QoS Metrics-In-the-loop for better Robot Navigation
(H2020 RobMoSys Project)

Thank you!

cristinav@unex.es

https://sites.google.com/view/cristina-vicente-chicote

https://www.researchgate.net/profile/Cristina-Vicente-Chicote

https://www.linkedin.com/in/cvicente/

@cvicentechicote