Model-based software development

Lecture XII.

Model-based developement

Dr. Semeráth Oszkár

Model-based developement

I. Development concepts

II. Development of critical systems

III. Feature modeling

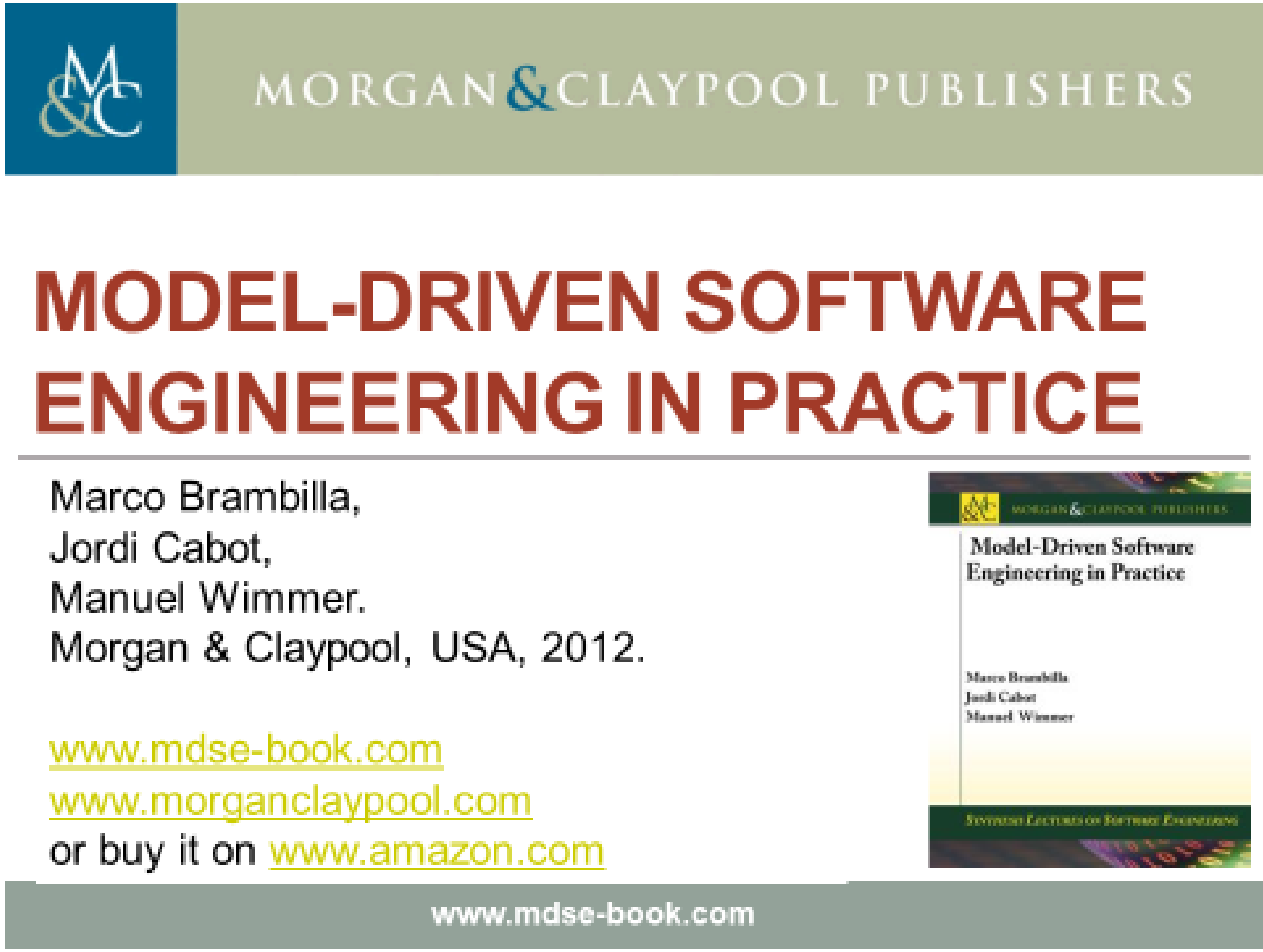
IV. Generative programming

Traditional motivations for MDSE

Principles and objectives

- What are the motivations for model-based development? What questions arise?

- We reused some materials with the permission of the authors



Traditional motivations for MDSE

Principles and objectives

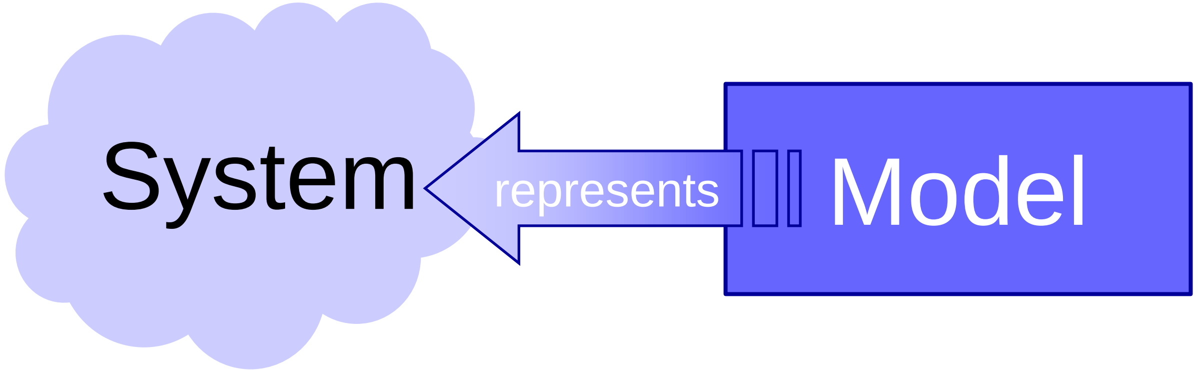
- **Abstraction** from specific realization technologies
  - Requires modeling languages, which do not hold specific concepts of realization technologies (e.g., Java EJB)
  - Improved **portability** of software to new/changing technologies – model once, build everywhere
  - **Interoperability** between different technologies can be automated (so called Technology Bridges)

- **Automated code generation** from abstract models
  - e.g., generation of Java-APIs, XML Schemas, etc. from UML
  - Requires expressive und precise models
  - Increased **productivity** and **efficiency** (models stay up-to-date)

- **Separate development** of application and infrastructure
  - Separation of application-code and infrastructure-code (e.g. Application Framework) increases **reusability**
  - **Flexible** development cycles as well as **different development roles possible**

Models

What is a model?



Mapping Feature	A model is based on an original (=system)
Reduction Feature	A model only reflects a (relevant) selection of the original‘s properties
Pragmatic Feature	A model needs to be usable in place of an original with respect to some purpose

Purposes:

- descriptive purposes
- prescriptive purposes

MDSE Equation

Models + Transformations = Software

Modeling Languages

- **Domain-Specific Languages (DSLs):** languages that are designed specifically for a certain domain or context
- DSLs have been largely used in computer science. Examples: HTML, Logo, VHDL, Mathematica, SQL

- **General Purpose Modeling Languages** (GPMLs, GMLs, or GPLs): languages that can be applied to any sector or domain for (software) modeling purposes
- The typical examples are: UML, Petri-nets, or state machines

Types of models

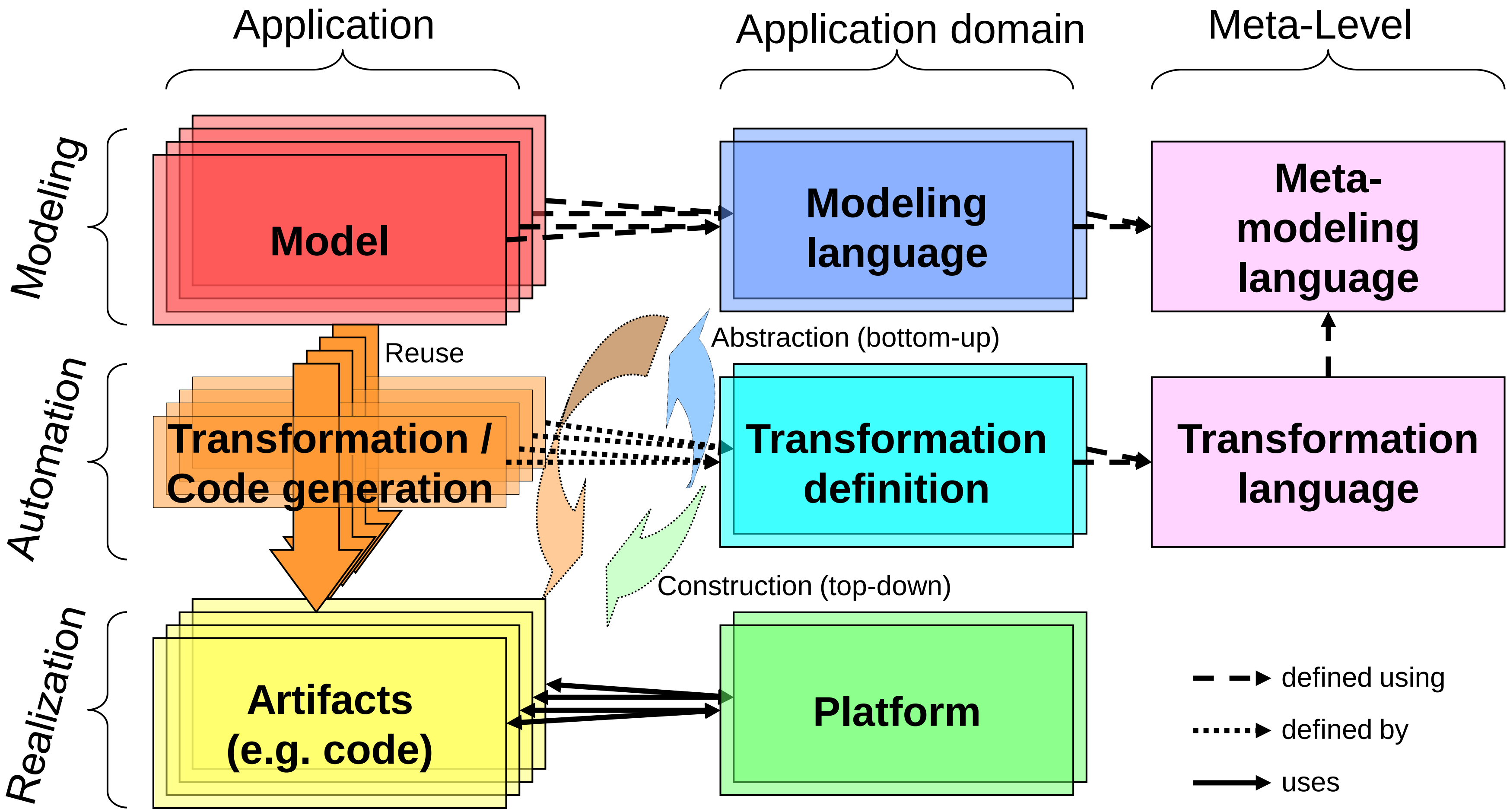
- **Static models:** Focus on the static aspects of the system in terms of managed data and of structural shape and architecture of the system.
- **Dynamic models:** Emphasize the dynamic behavior of the system by showing the execution
- **Runtime models:** Describe the state of the system during operation.

- Just think about UML!

Usage / Purpose:
- Traceability Models:
- Execution Trace Models
- Analysis Models
- Simulation Models

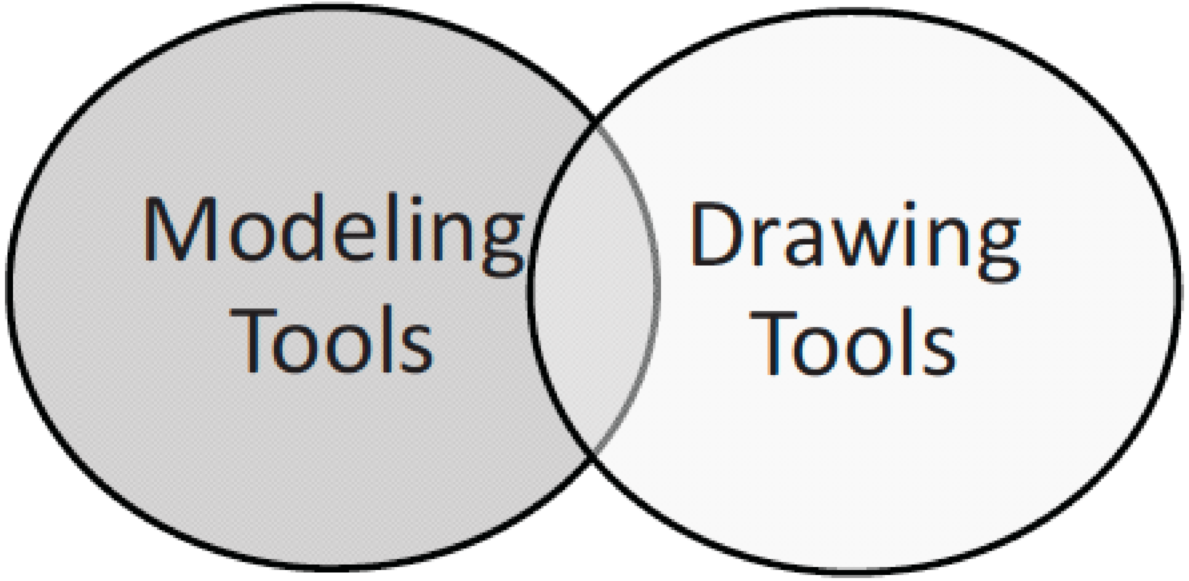
Concepts

Model Engineering basic architecture



Tool support

- Drawing vs. modeling

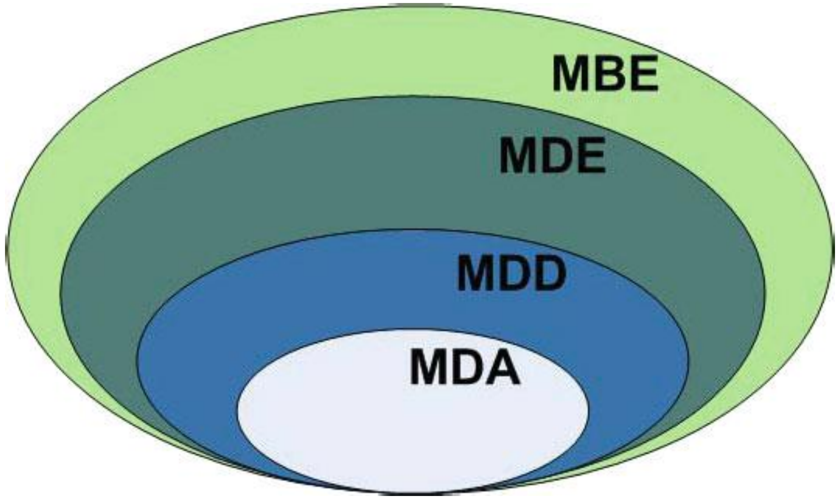


Concepts

Consequences or Preconditions

- Modified **development process**
  - Two levels of development – application and infrastructure
    - Infrastructure development involves modeling language, platform (e.g. framework) and transformation definition
    - Application development only involves modeling – efficient reuse of the infrastructure(s)
  - Strongly simplified application development
    - Automatic code generation replaces programmer
    - Working on the code level (implementation, testing, maintenance) becomes unnecessary
    - *Under which conditions is this realistic … or just futuristic?*
- New **development tools**
  - Tools for language definition, in particular meta modeling
  - Editor and engine for model transformations
  - Customizable tools like model editors, repositories, simulation, verification, and testing tools

The MD* Jungle of Acronyms



- **Model-Driven Development (MDD)** is a development paradigm that uses models as the primary artifact of the development process.
- **Model-Driven Architecture (MDA)** is the particular vision of MDD proposed by the Object Management Group (OMG)
- **Model-Driven Engineering (MDE)** is a superset of MDD because it goes beyond of the pure development
- **Model-Based Engineering** (or “model-based development”) (**MBE**) is a softer version of ME, where models do not “drive” the process.

The MDA Approach

Goals

- **Interoperability** through Platform Independent Models
  - Standardization initiative of the Object Management Group (**OMG**), based on OMG Standards, particularly **UML**
  - Counterpart to CORBA on the modeling level: interoperability between different platforms
  - Applications which can be installed on different platforms → portability, no problems with changing technologies, integration of different platforms, etc.
- **Modifications to the basic architecture**
  - Segmentation of the model level
    - **Platform Independent** Models (PIM): valid for a set of (similar) platforms
    - **Platform Specific** Models (PSM): special adjustments for one specific platform
  - Requires model-to-model transformation (PIM-PSM; compare QVT) and model-to-code transformation (PSM-Code)
  - Platform development is not taken into consideration – in general industry standards like J2EE, .NET, CORBA are considered as platforms

[www.omg.org/mda/]

Modelling Levels

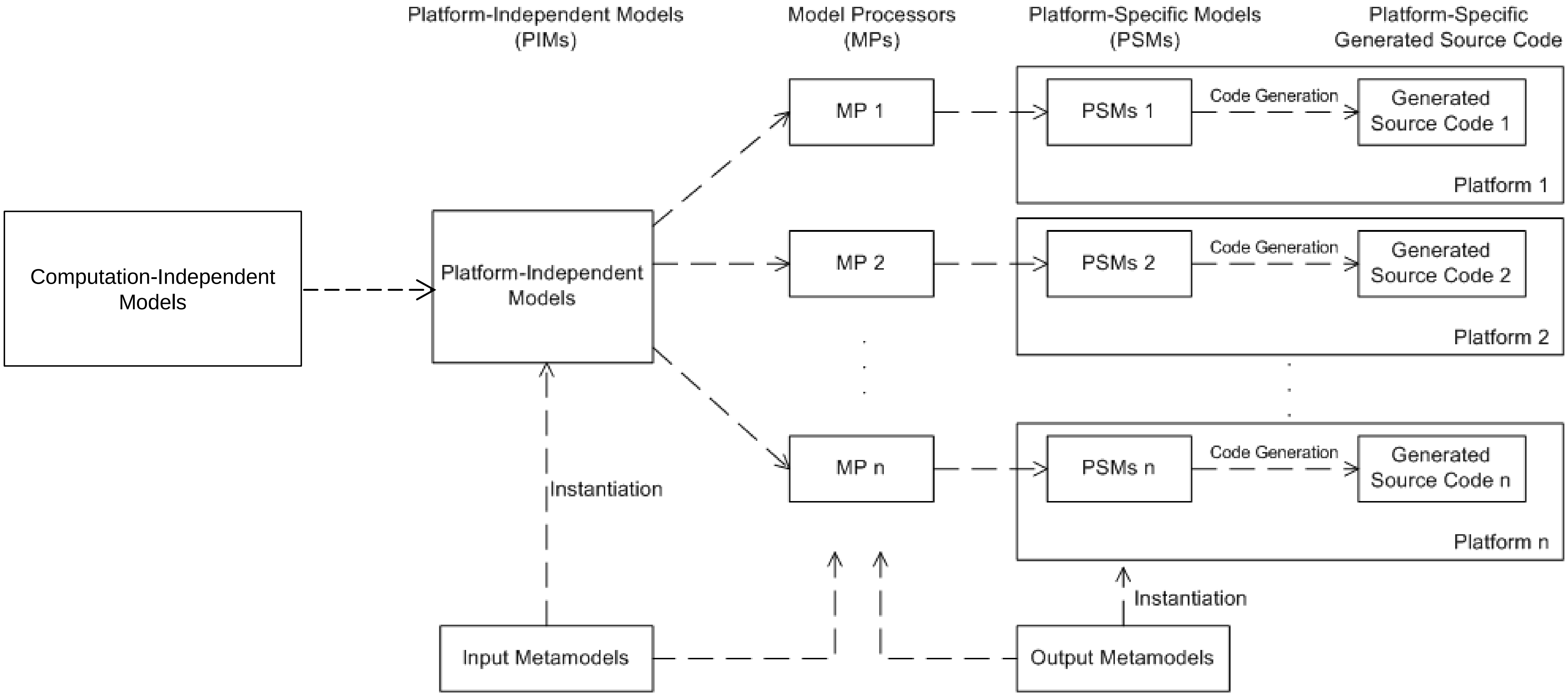
CIM, PIM, PSM

- **Computation independent models (CIM):** describe requirements and needs at a very abstract level, without any reference to implementation aspects (e.g., description of user requirements or business objectives);

- **Platform independent models (PIM):** define the behavior of the systems in terms of stored data and performed algorithms, without any technical or technological details;

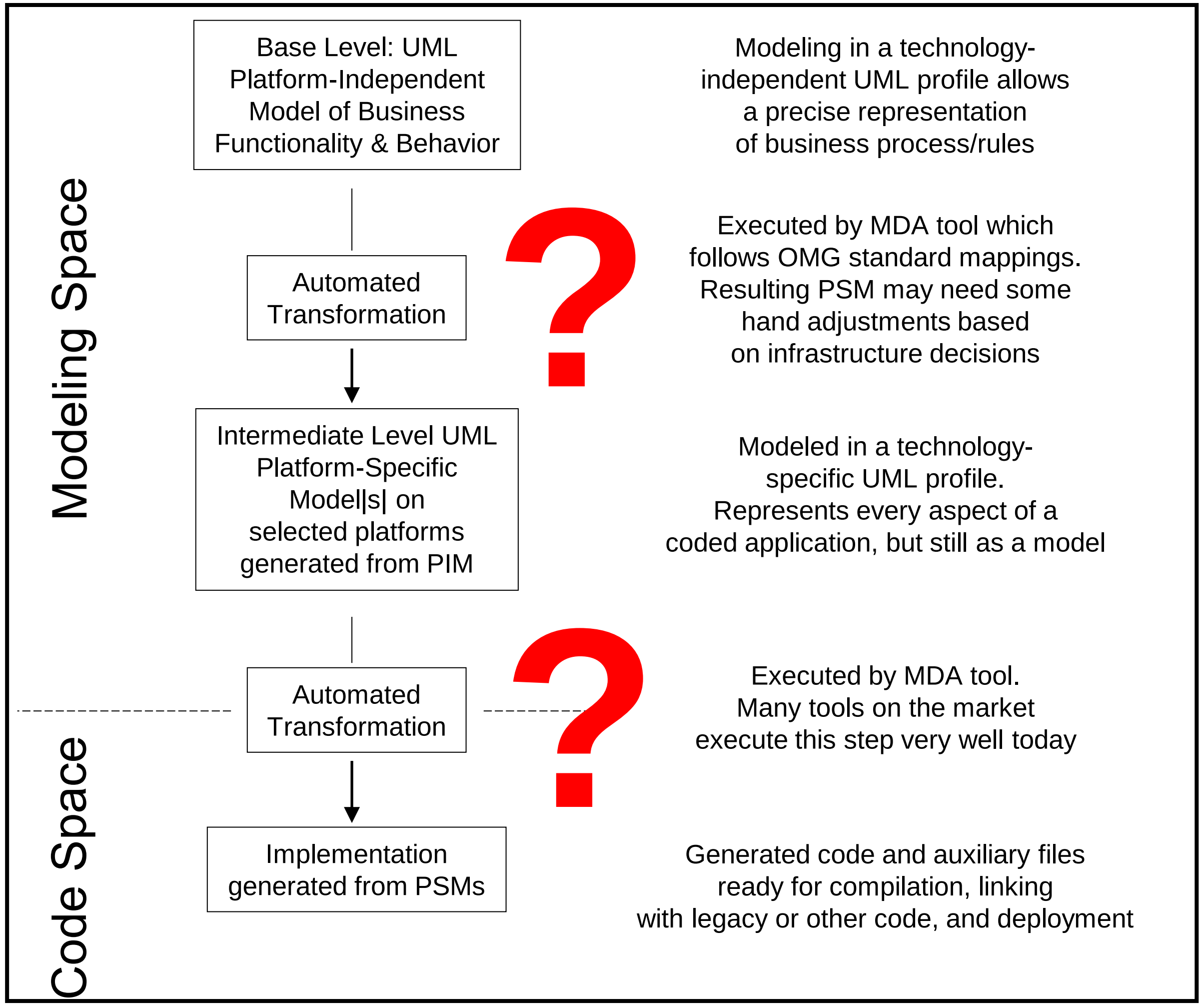
- **Platform-specific models (PSM):** define all the technological aspects in detail.

How do we picture this?



The MDA Approach

MDA development cycle



Model-based developement

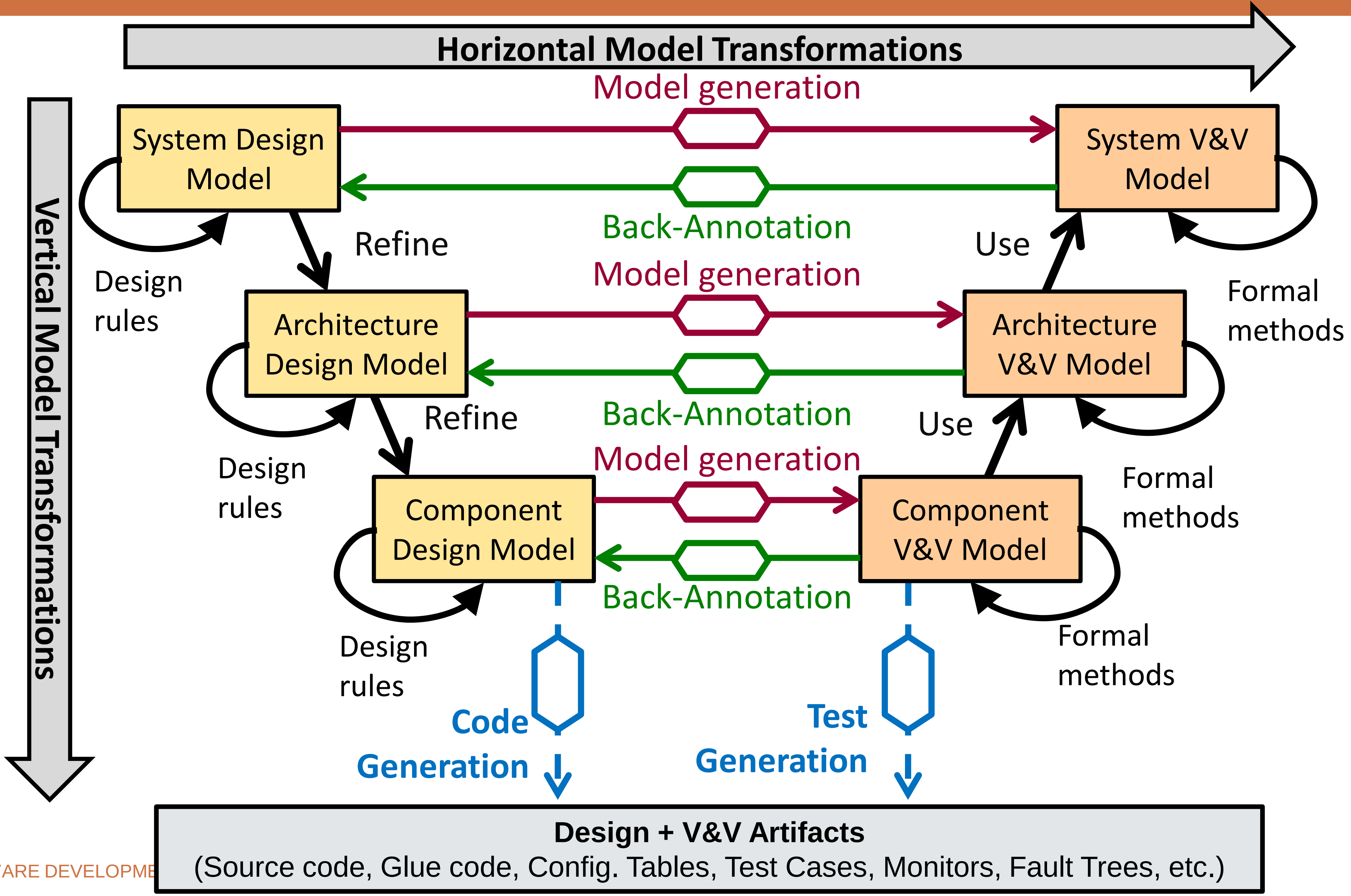
I. Development concepts

II. Development of critical systems

III. Feature modeling

IV. Generative programming

Models and Transformations in Critical Systems Development



Development Process for Critical Systems

Unique Development Process (Traditional V-Model)

Critical Systems Design

- requires a certification process
- to develop justified evidence
- that the system is free of flaws

Software Tool Qualification

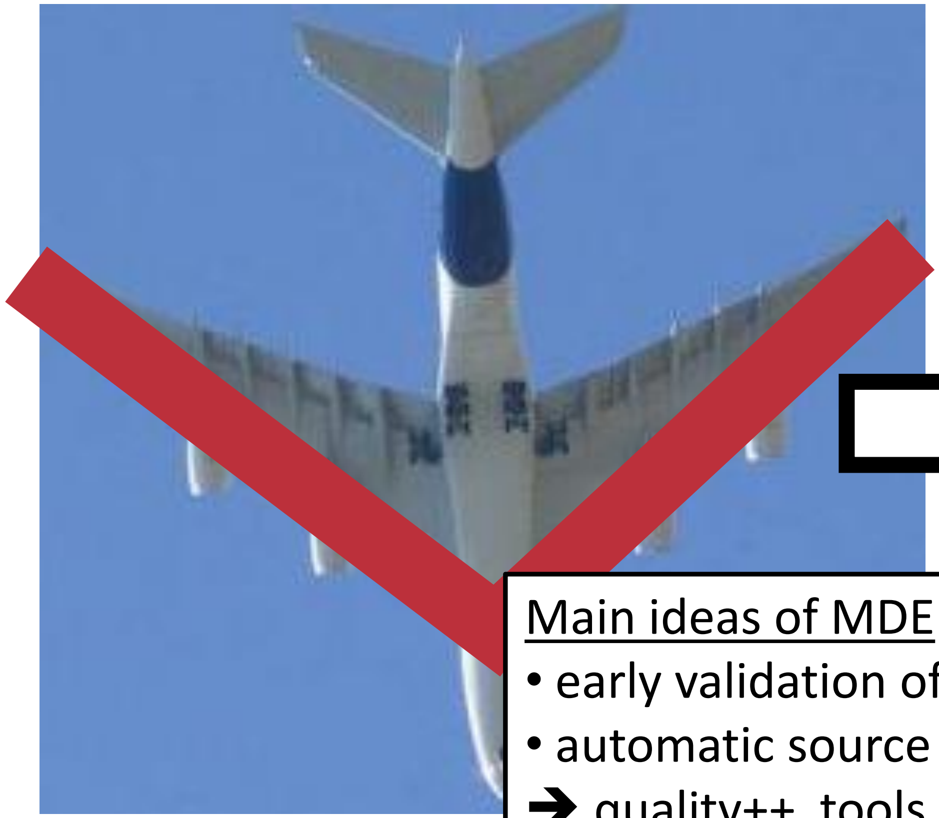
- obtain certification credit
- for a software tool
- used in critical system design

Innovative Tool → Better System

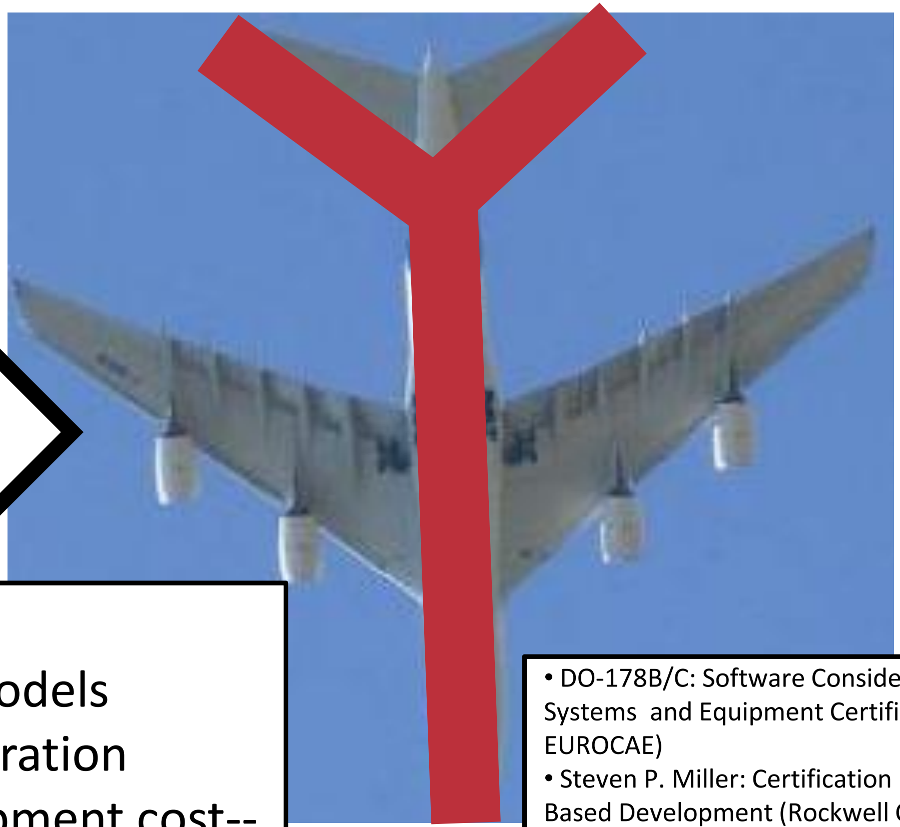
Qualified Tool → Certified Output

Development Process for Critical Systems

Unique Development Process (Traditional V-Model)



Model-Driven Engineering (Y-Model)



Main ideas of MDE

- early validation of system models
- automatic source code generation

➔ quality++ tools ++ development cost--

- DO-178B/C: Software Considerations in Airborne Systems and Equipment Certification (RTCA, EUROCAE)
- Steven P. Miller: Certification Issues in Model Based Development (Rockwell Collins)

Model-based developement

- I. Development concepts
- II. Development of critical systems
- **III. Feature modeling**
- IV. Generative programming

Feature modeling – introduction

- Differences between elements of a product family
  - Mobile phones
    - Display type
    - I/O interfaces
  - Automotive
    - Number of doors
    - Engine type

- Domain specific language for combining differences: feature modeling

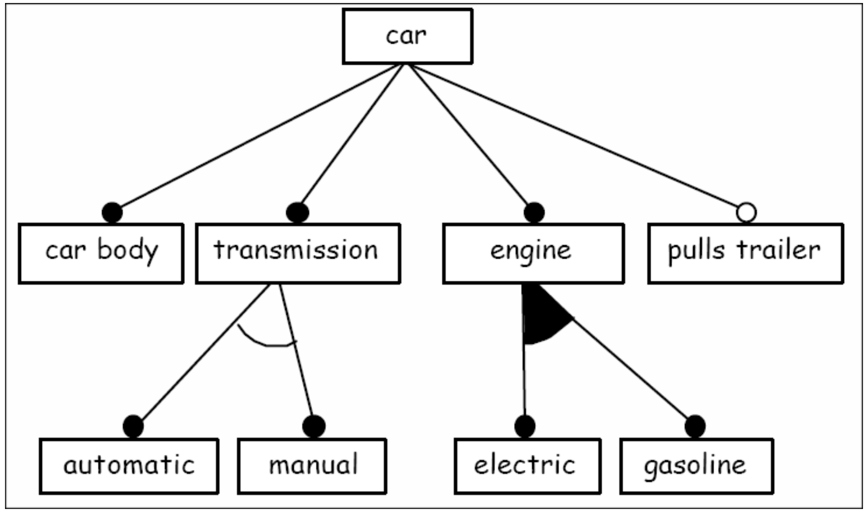
Feature modeling

- Feature model
  - *an implementation-independent, concise description of the different domain variants*
  - differences between specific product instances
  - product family configuration options

- Key: reuse
- Helps to avoid:
  - Missing an important feature/variation
  - Adding unnecessary features / variations

Feature modeling

- Model elements
  - Nodes
  - Directional edges
  - Marking at edges
- Root element: concept
- Features (feature node)



- Configuration: subset of features
- Configuration must follow certain rules!

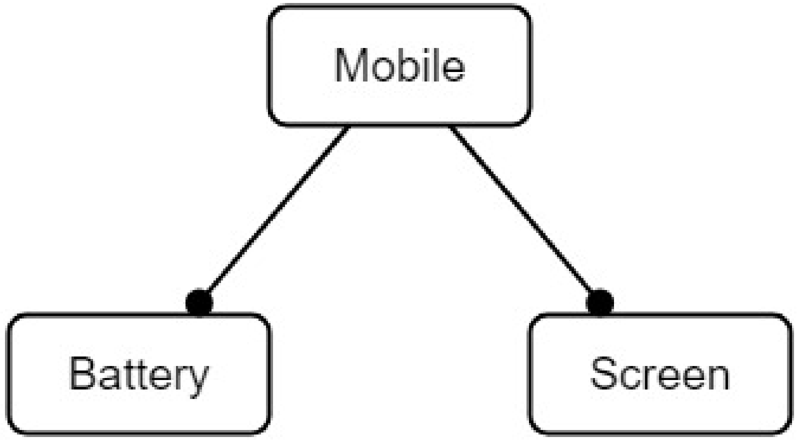
Feature modeling

- Focus on the feature
  - *Part of knowledge, building block of the concept*
  - Helps to find similarities and differences between products, product groups
  - Useful when there are several versions of the same product

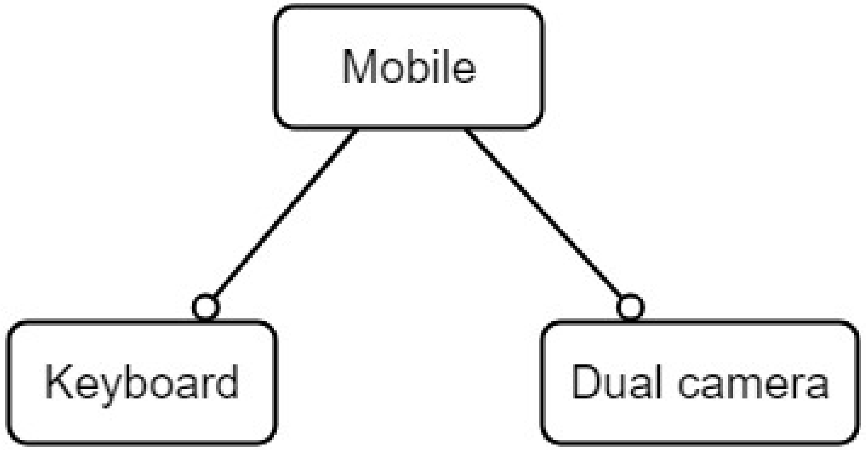
- https://modeling-languages.com/analysis-of-feature-models/

Feature modeling

- Mandatory feature

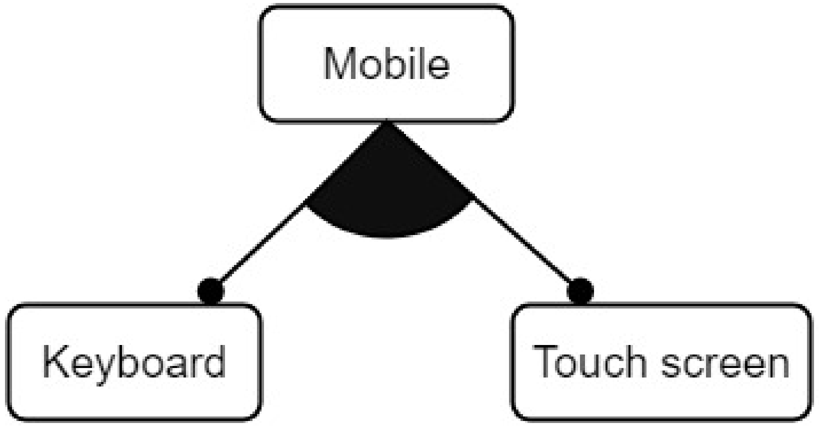


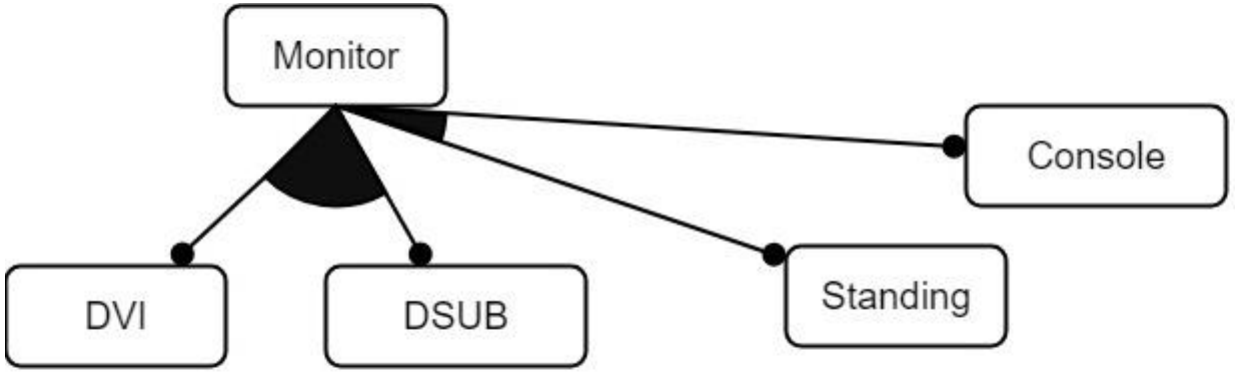
- Optional feature



Feature modeling

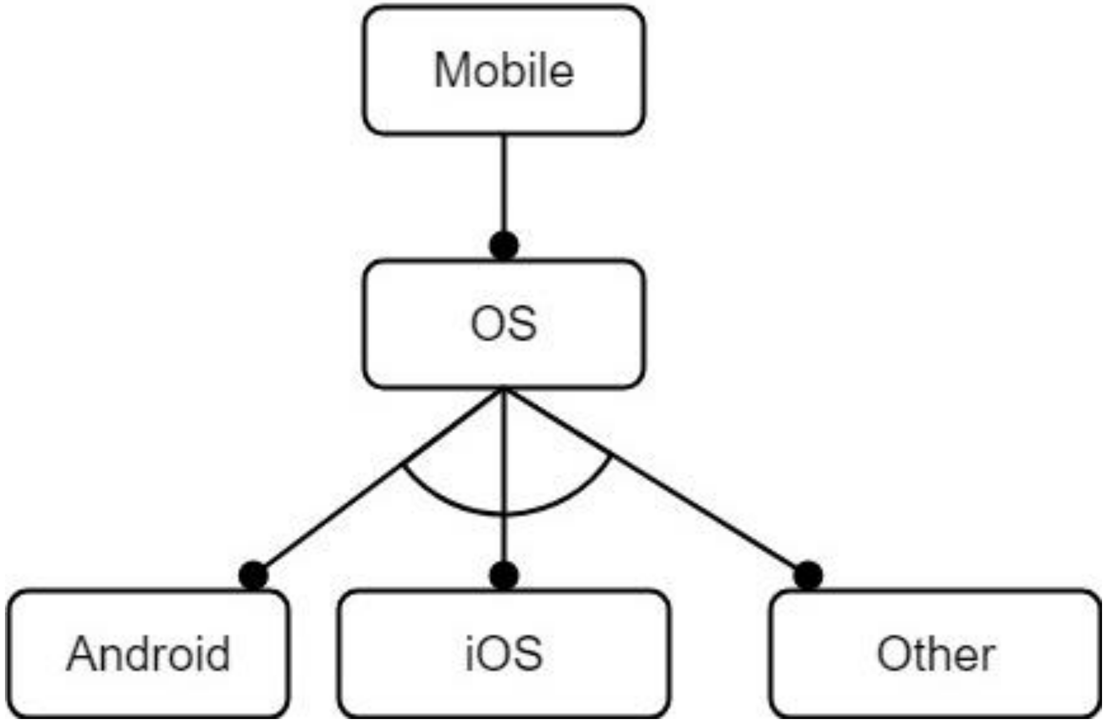
- „Or” relationship (at least 1)





Feature modeling

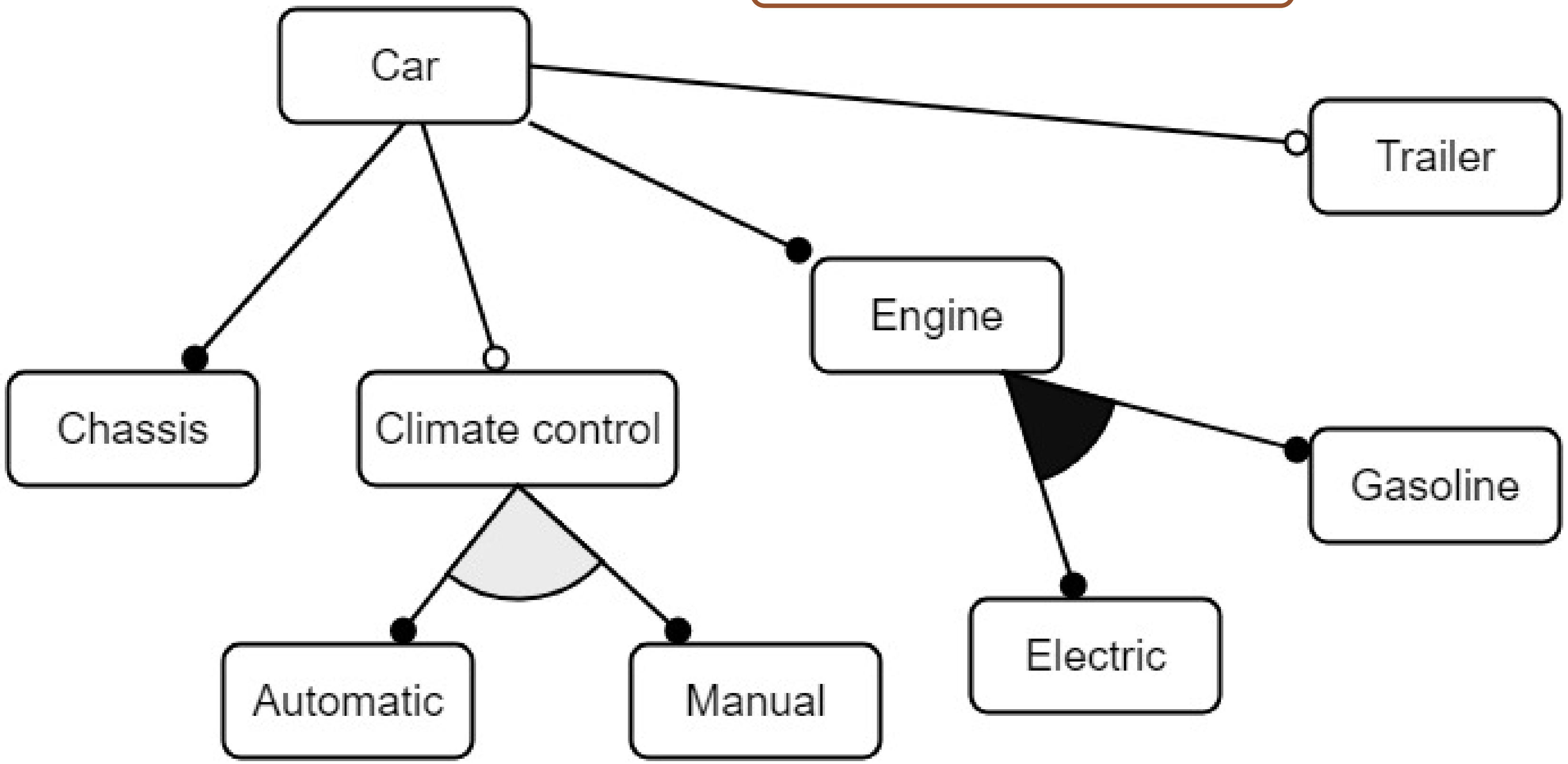
- Alternative features (one of the elements)



Feature modeling

- Example: car

+ Logical constraints

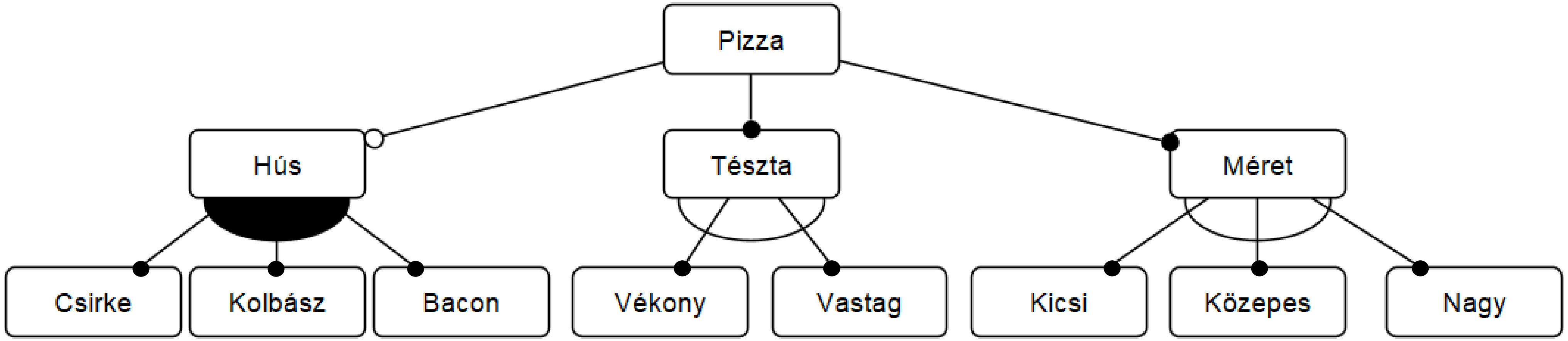


Feature modeling – Example

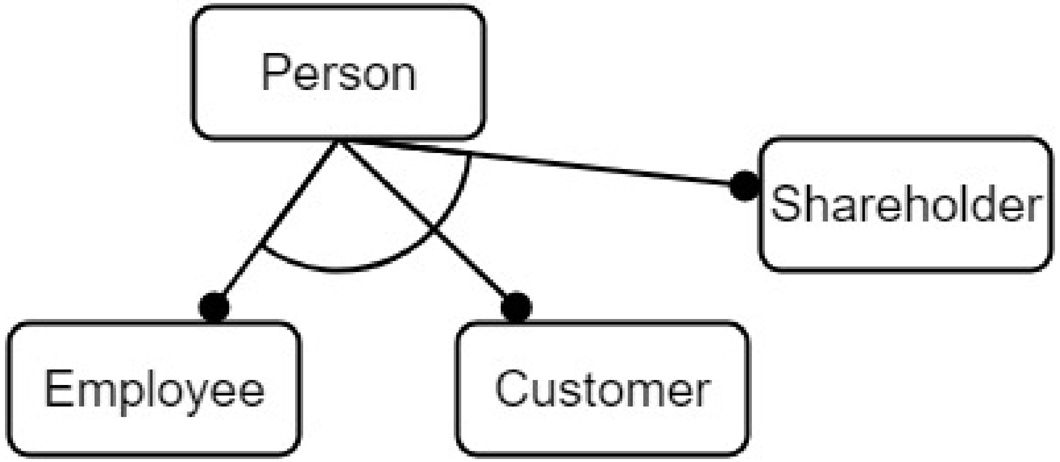
- Create a feature model for the following task: making a pizza. The model should allow you to specify, among other things, meat (chicken, sausage, bacon), additional toppings (tomato, onion, pepper), dough type (traditional, light), size (small, medium, large), etc. The model should include optional, mandatory and exclusive (OR) features. Briefly justify the structure of the model in text.

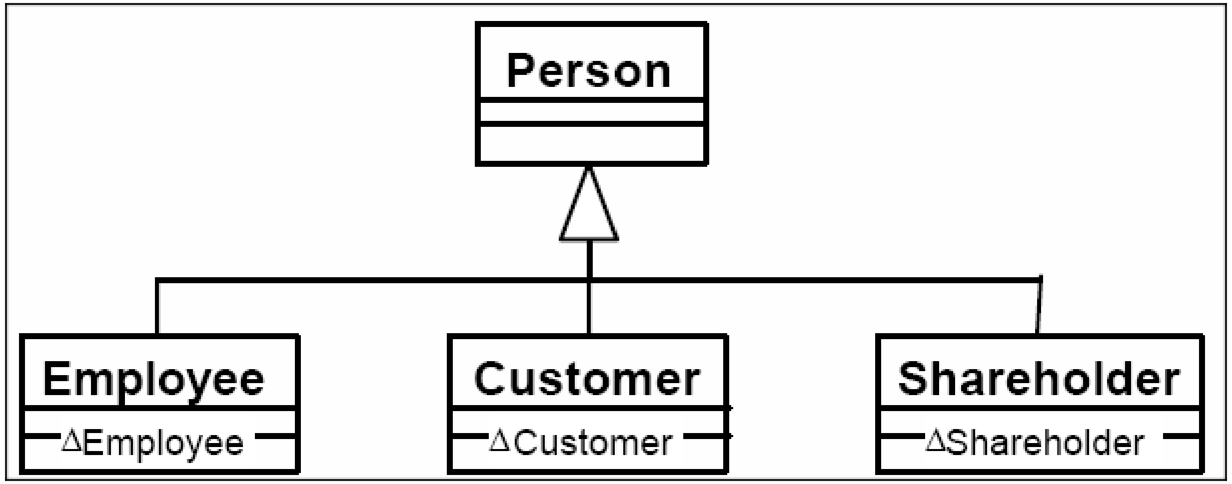
Feature modeling – Example – Solution

- Create a feature model for the following task: making a pizza. The model should allow you to specify, among other things, meat (chicken, sausage, bacon), additional toppings (tomato, onion, pepper), dough type (traditional, light), size (small, medium, large), etc. The model should include optional, mandatory and exclusive (OR) features. Briefly justify the structure of the model in text.

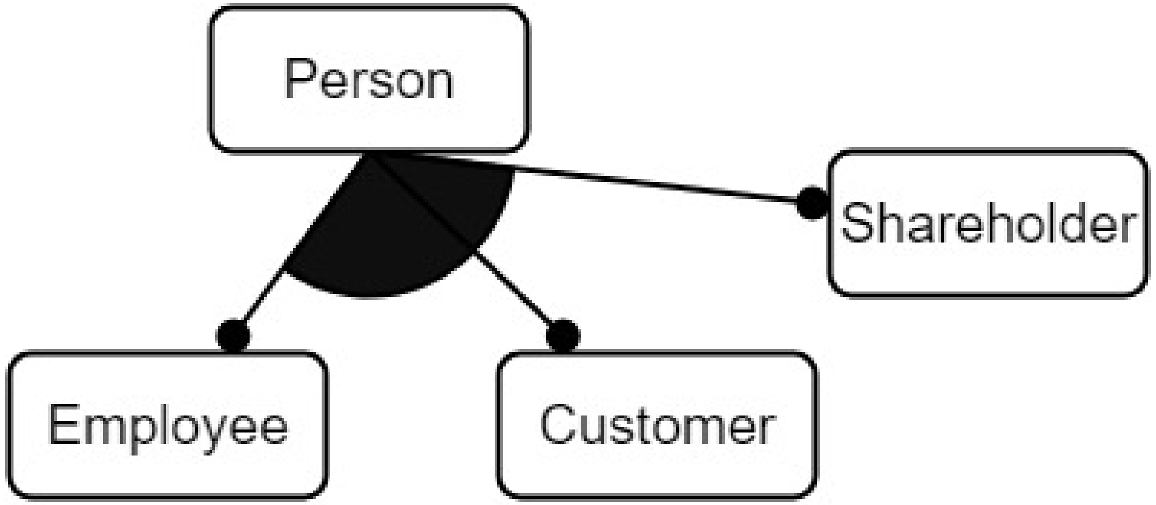


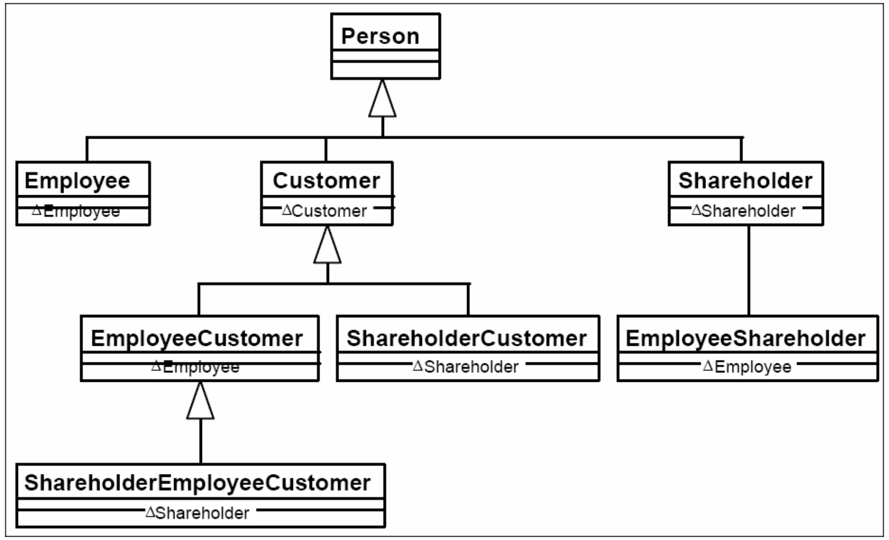
Feature modeling – code generation





Feature modeling – code generation

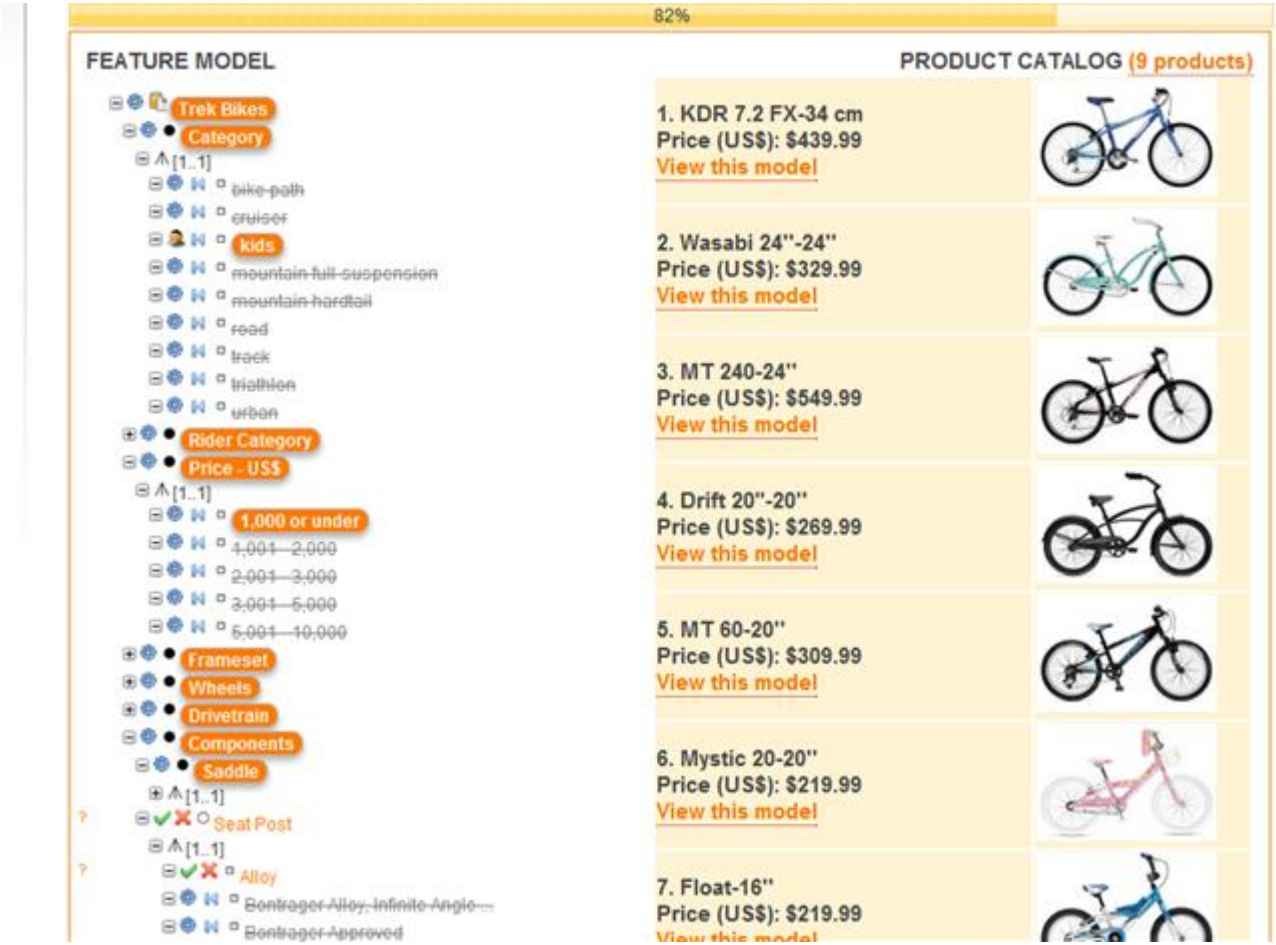




Feature modeling in practice

- Target application
  - Web catalogue of products that match the feature model
  - Feature model-based search in the catalogue
- Generation based on the model
  - Web application
  - Database table definitions

Feature modeling in practice



- Modeling features
- Configuration selection
- Code generation
- Solving the task

- #Configuration > #Product

Marcilio Mendonca, Andrzej Wąsowski, and Krzysztof Czarnecki. 2009. ***SAT-based analysis of feature models is easy****. In Proceedings of the 13th International Software Product Line Conference (SPLC '09). Carnegie Mellon University, USA, 231–240.*

Model-based developement

I. Development concepts

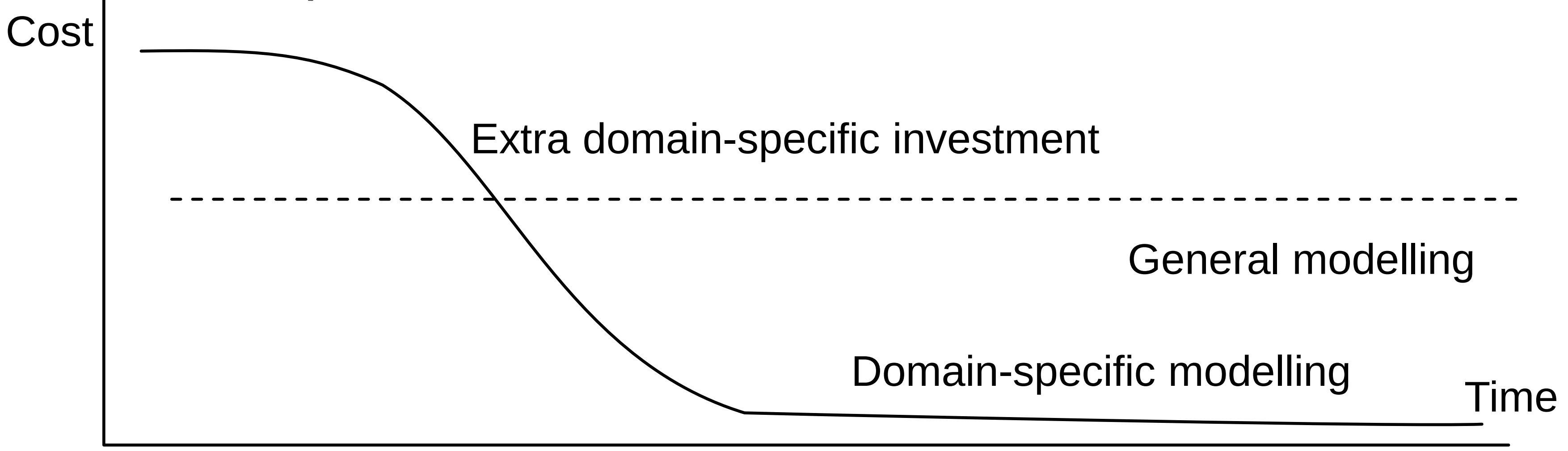
II. Development of critical systems

III. Feature modeling

IV. Generative programming

Generative programming

- Programming methodology based on automatic source code generation
- Can be paralleled with component-based software development and product line design
- Reusable termék
- Not one-off development, continuous evolution



Generative programming

- Generative paradigm
  - Operation: modelling language + generators
  - Worth it for repeated use
- Code generation
  - No universal DSL compilers (like C compilers)
  - Often DSL and generator are developed in the same place
    - Fast development, fine tuning possible
    - Potential for errors

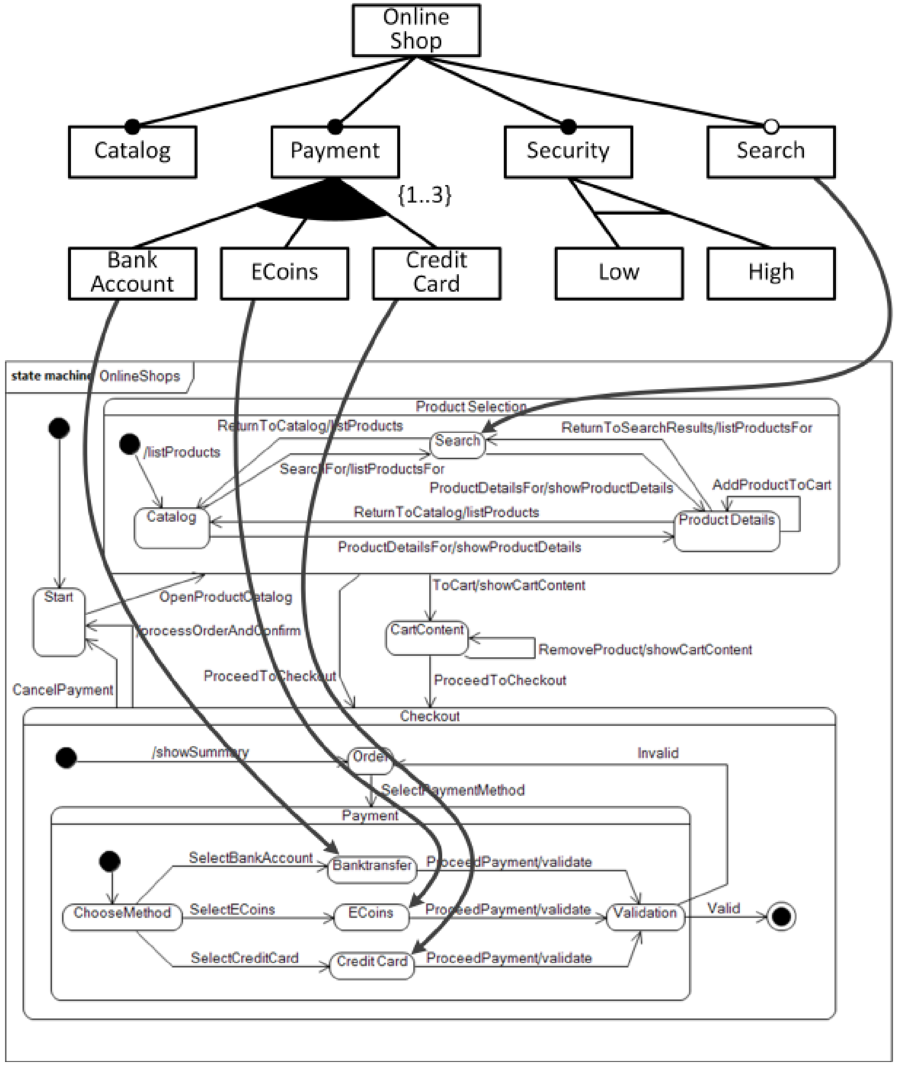
Generating an application

- Typical model processing: generating an application
  - What we need
  - Model of a domain specific language
  - Generator
  - Keretrendszer (pl. osztálykönyvtár)
- Challenges:
  - Too detailed/generic language → small abstraction level jump → small benefit from generator
  - Domain does not fit → complex generator
    - validation + due to added info
    - Sign: developers build the model in a way that the generator will accept

Mapping features to model elements

- What to do when you need a model, not code?
- Mapping features to model elements
- 1 combination = 1 model
- $\sum konfiguration$ = 150% model
- The 150% model is not necessarily a regular model

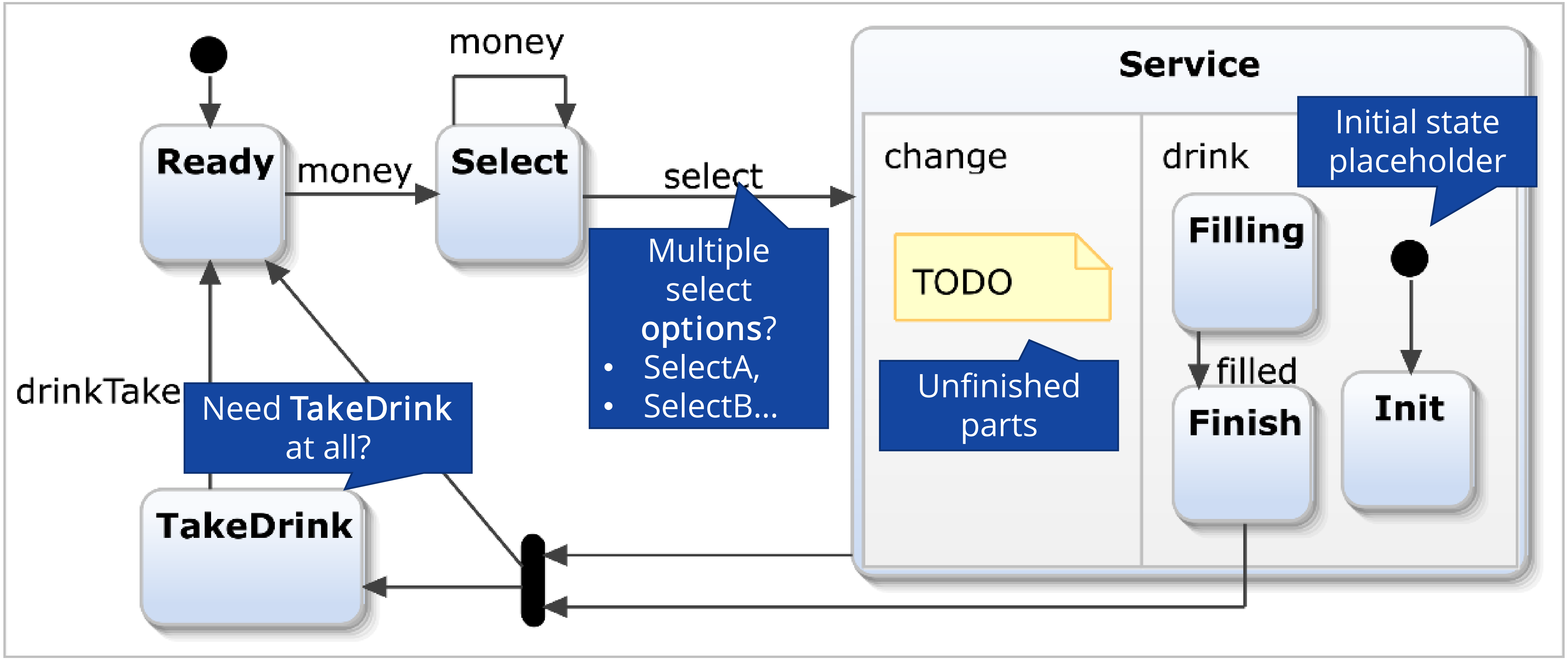
Stephan Weißleder, Hartmut Lackner: Top-Down and Bottom-Up Approach for Model-Based Testing of Product Lines



Partial Modeling

MAVO

Example: Unfinished models



Motivation

- Early phase of development → high uncertainty in the models
- Editor forces the developer to work with complete models

Missing ⇔ Undecided / Uncertain / Unknown

Model refinement ⇔ Model rewriting

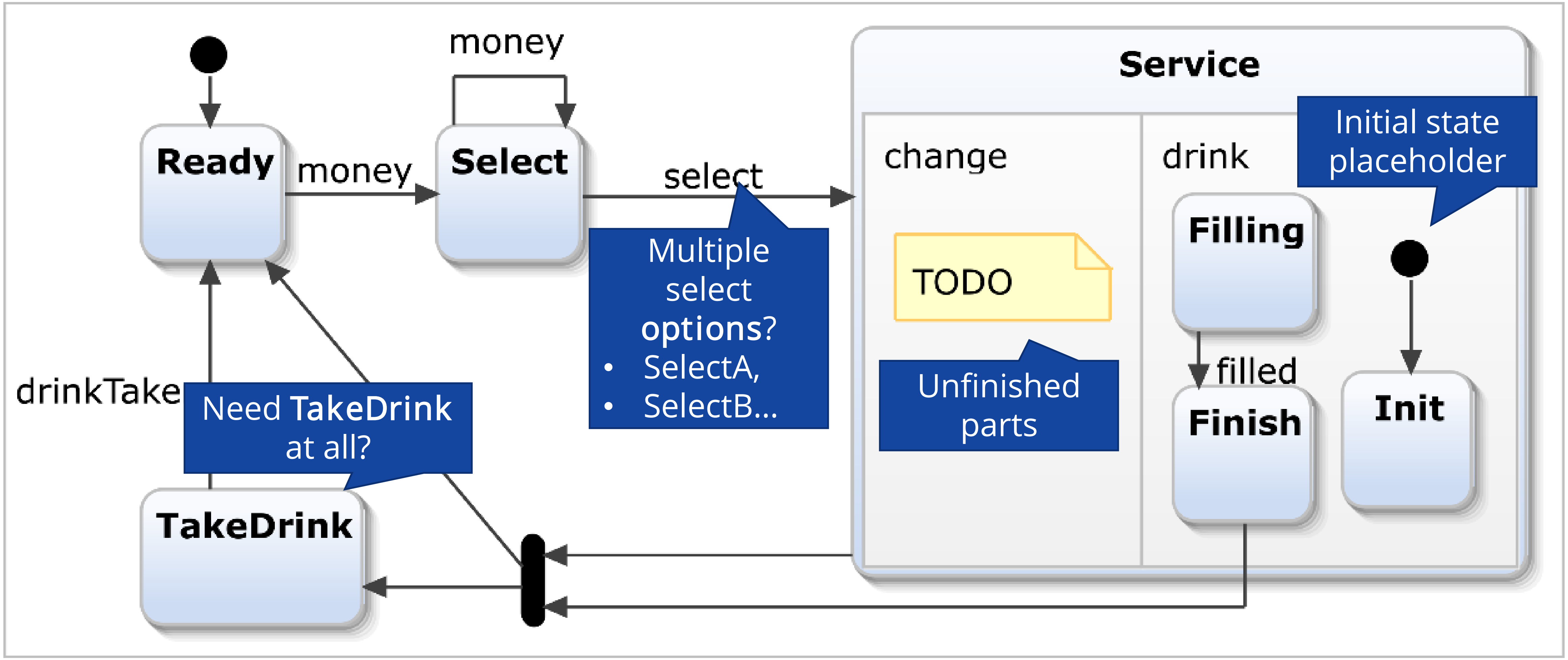
- **Issues:**
  - Forces the developer to make premature decisions
  - No way to list / document design alternatives
  - Editor mixes: invalid ⇔ unfinished
- **Clarify the semantics of missing elements**

Partial Modeling

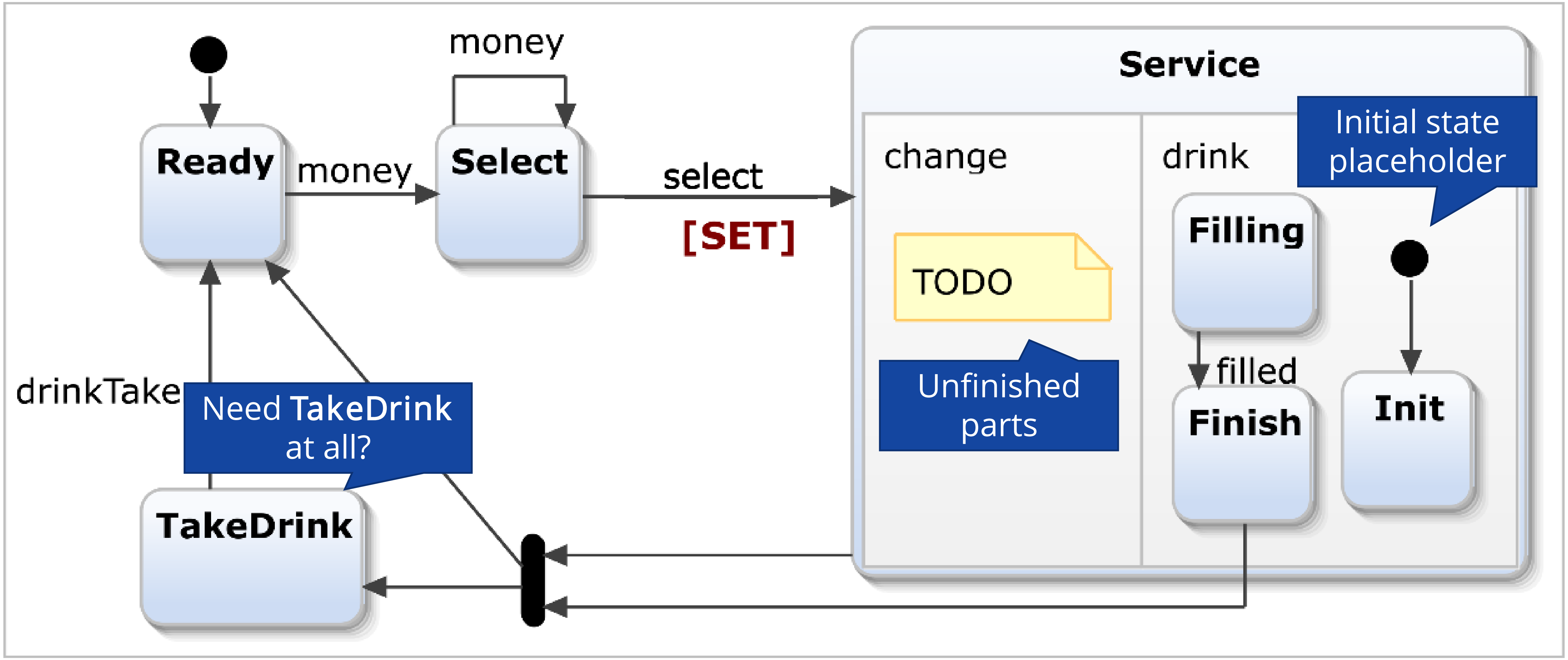
- Generic technique to explicitly represent uncertainty in models
  - Generic: works for every metamodel
  - Explicitly represent: uncertainty = model element
  - In Models: The uncertainty is attached to the models
- **MAVO:** practical way to annotate model with uncertainty
  - **M**ay: elements can be omitted
  - **A**bstract (Set): representing sets of elements
  - **V**ar: elements that can be merged
  - **O**pen: new elements can be added
- Automation: generate alternatives, check all alternatives

Michalis Famelis, Rick Salay, and Marsha Chechik. Partial models: towards modeling and reasoning with uncertainty. In: Proceedings of the 34th International Conference on Software Engineering, pp. 573–583. IEEE Press, 2012.

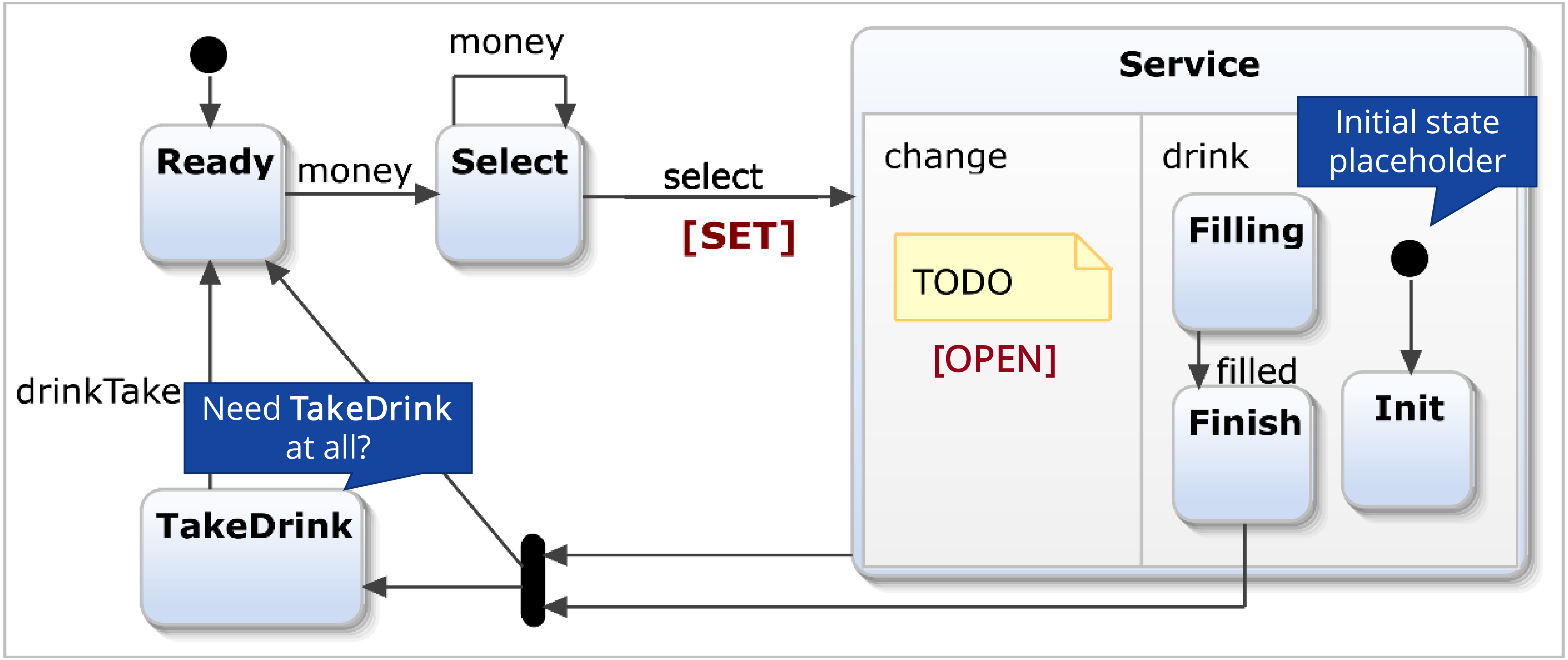
Example: Unfinished models with MAVO



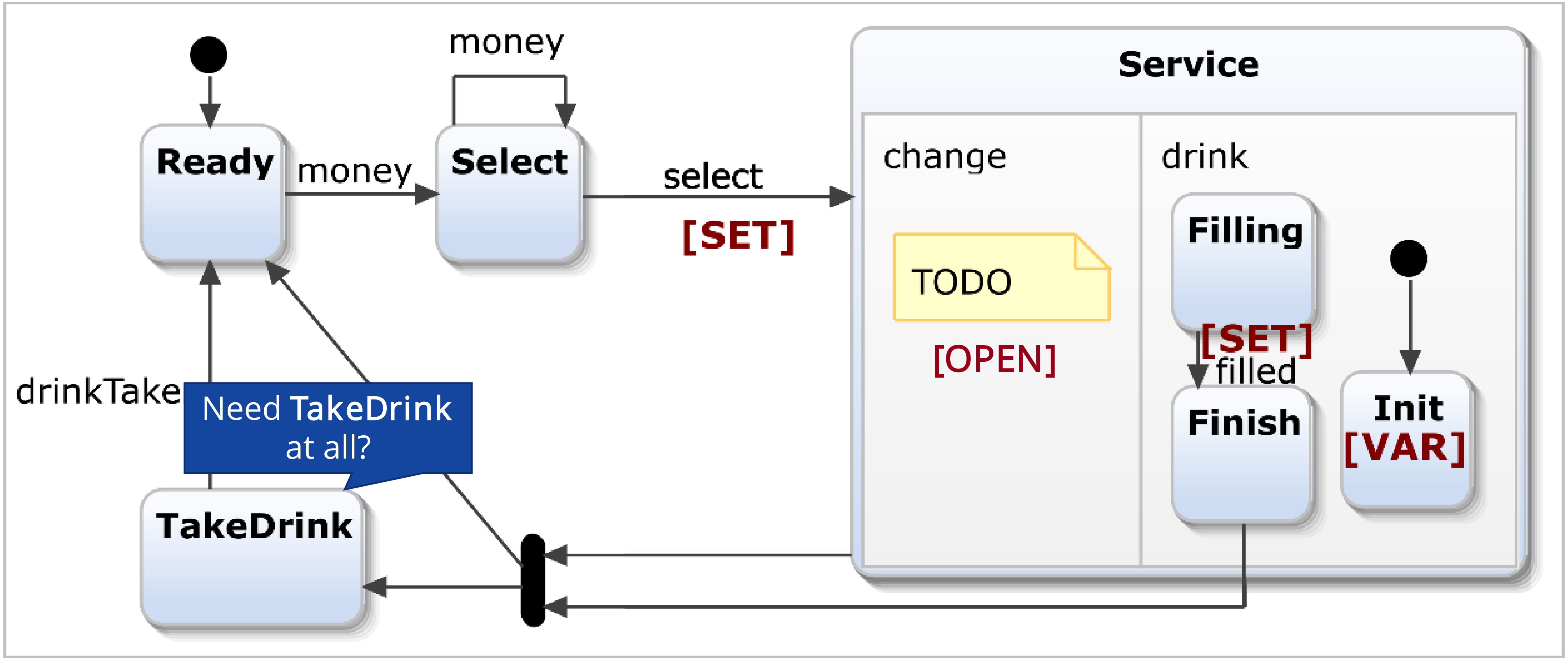
Example: Unfinished models with MAVO



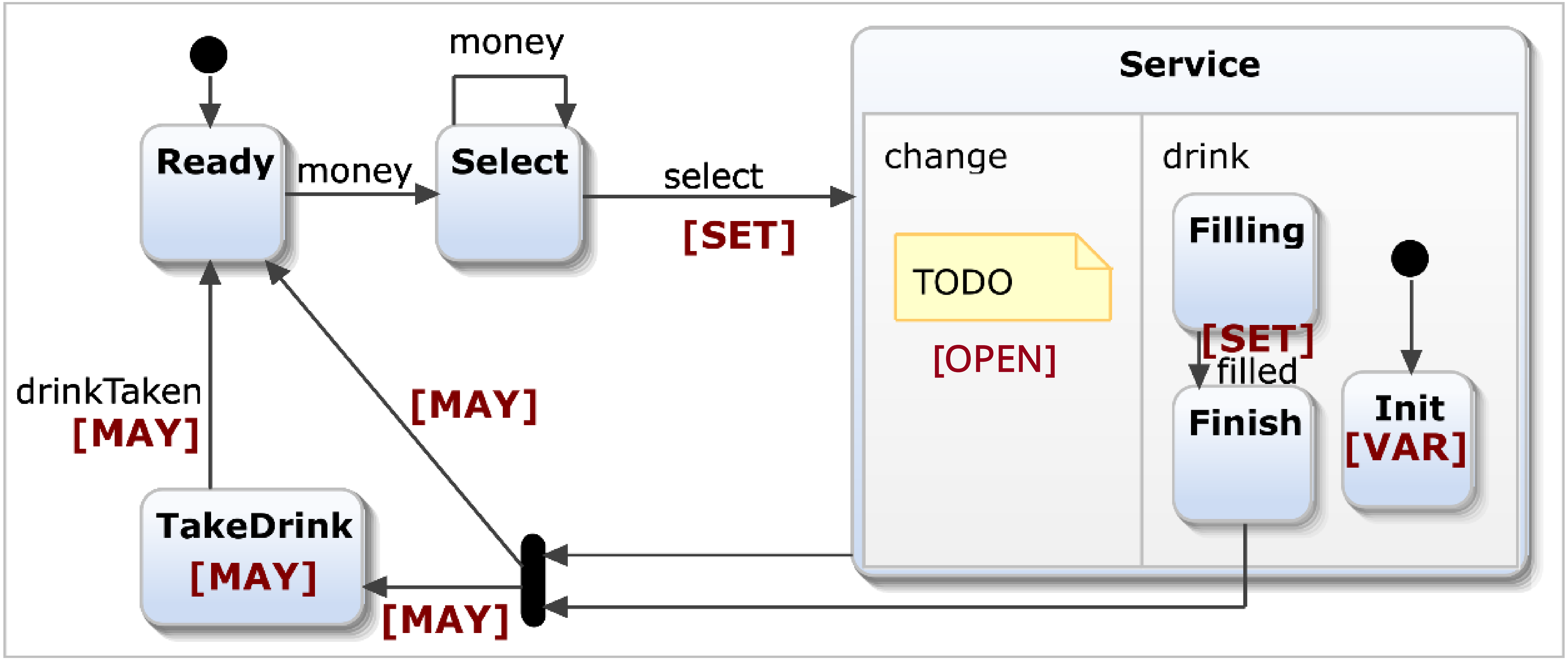
Example: Unfinished models with MAVO



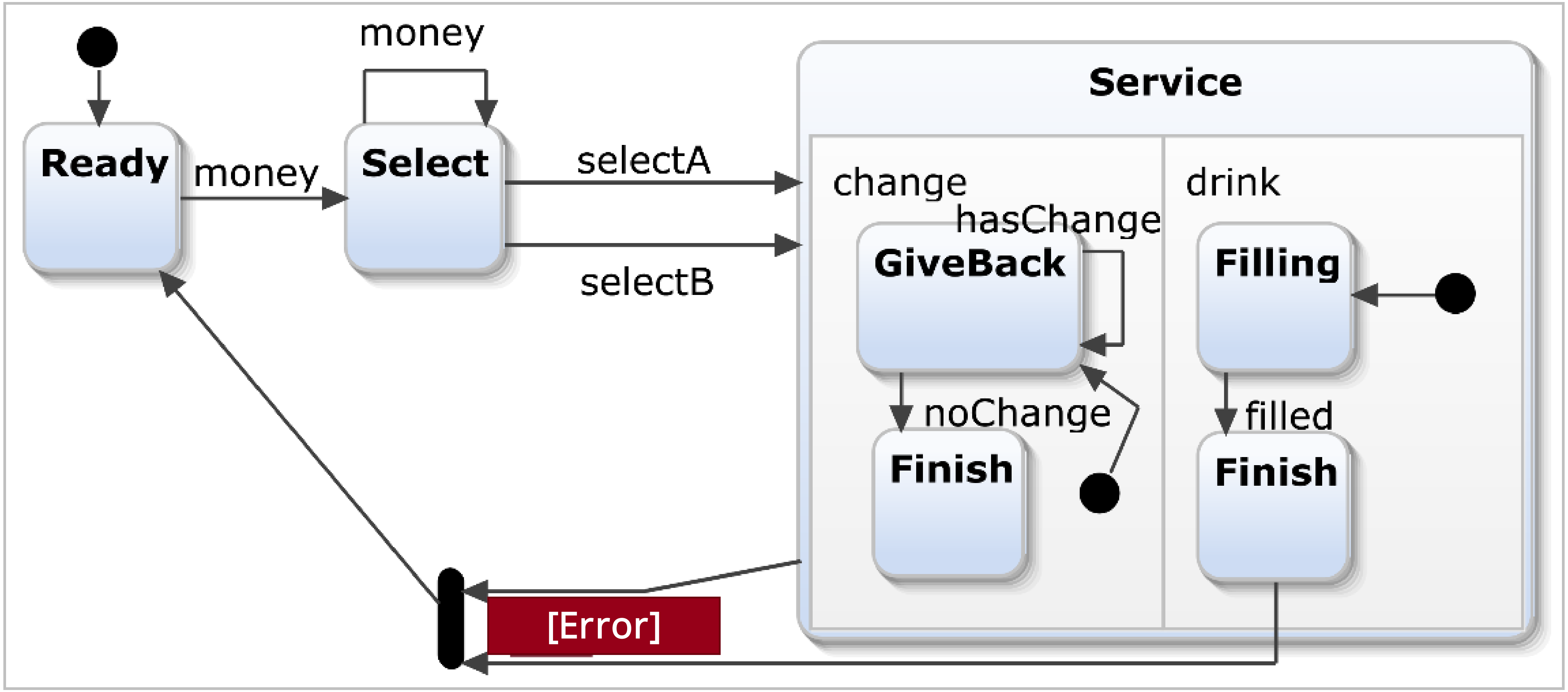
Example: Unfinished models with MAVO



Example: Unfinished models with MAVO



Example: Example concretization



MAVO Modeling Summary

- Partial modeling captures the uncertainty of models
- 1 partial model = set of complete model
- MAVO: framework for uncertainty annotation + tooling

- Semantics of missing vs unfinished

Partial models: Towards modeling and reasoning with uncertainty

M Famelis, R Salay, M Chechik

2012 34th International Conference on Software Engineering (ICSE), 573-583

4-valued partial models

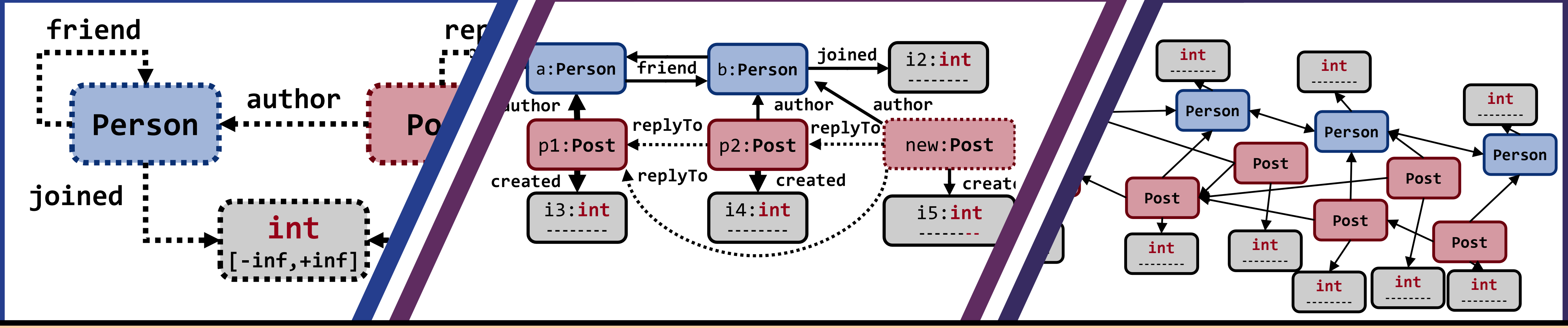
Partial Models

Intermediate state

Abstract models
(Metamodel + Constraints)

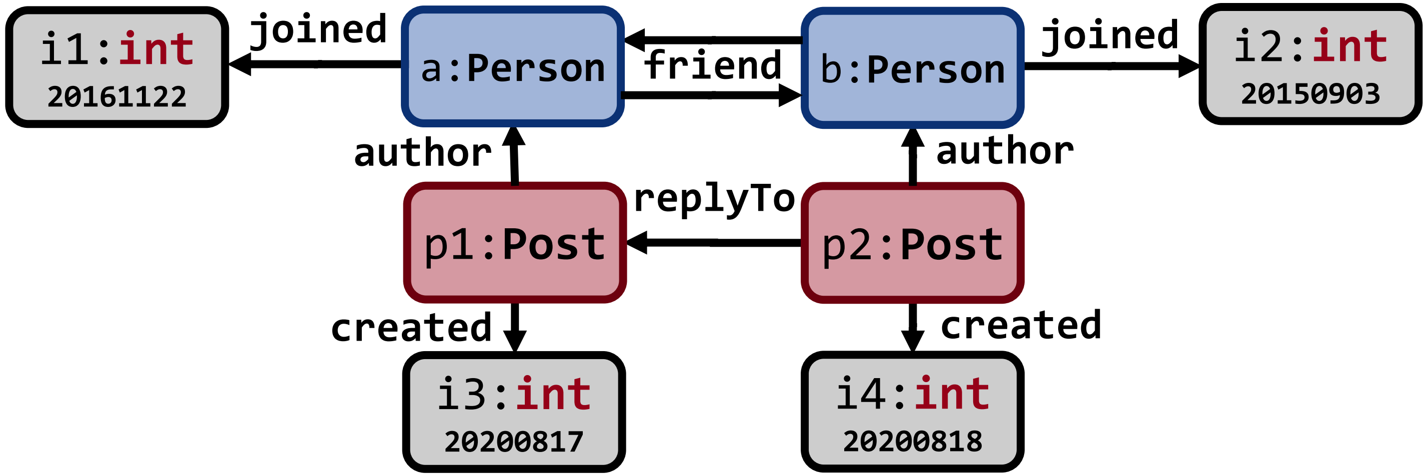
Partial models:
explicitly represent
uncertainty in models.

Concrete models
(Labelled graphs)

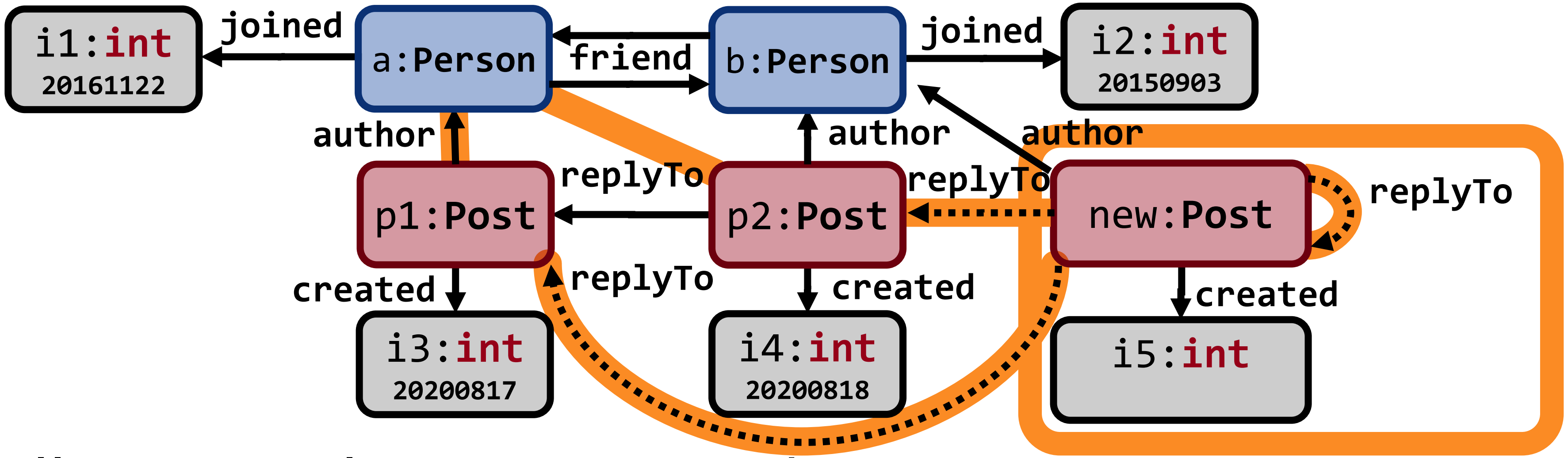


Model generation: exploration process that gradually reduces uncertainty

Partial Modeling: 4-valued logic

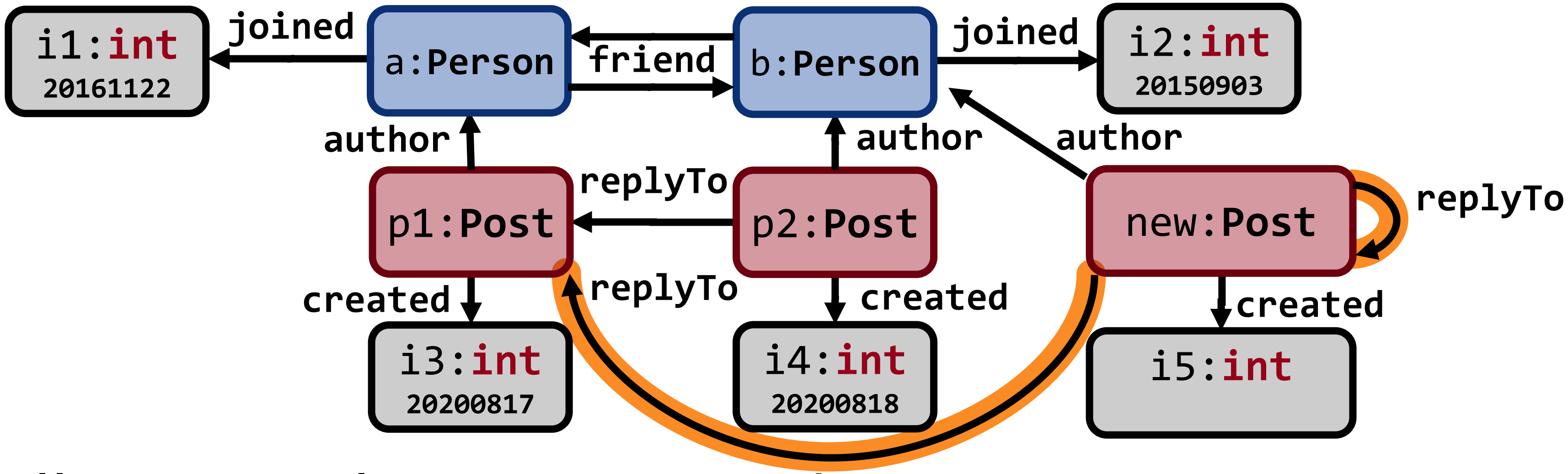


Partial Modeling: 4-valued logic



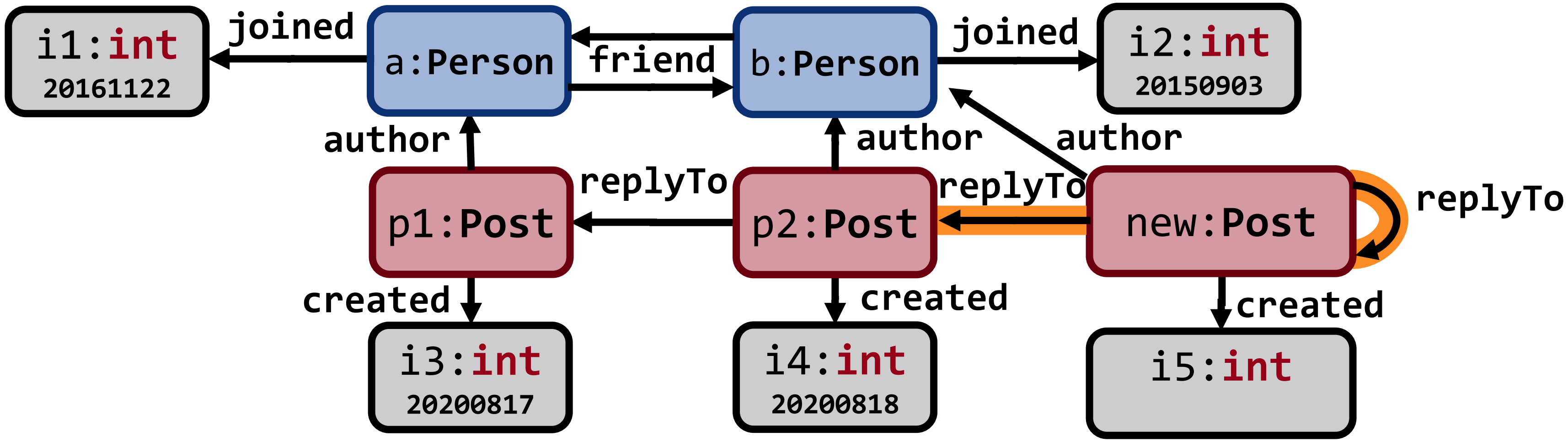
- Represent all potential extension with uncertainty
- Logic abstraction: true | false | unknown | error

Partial Modeling: 4-valued logic



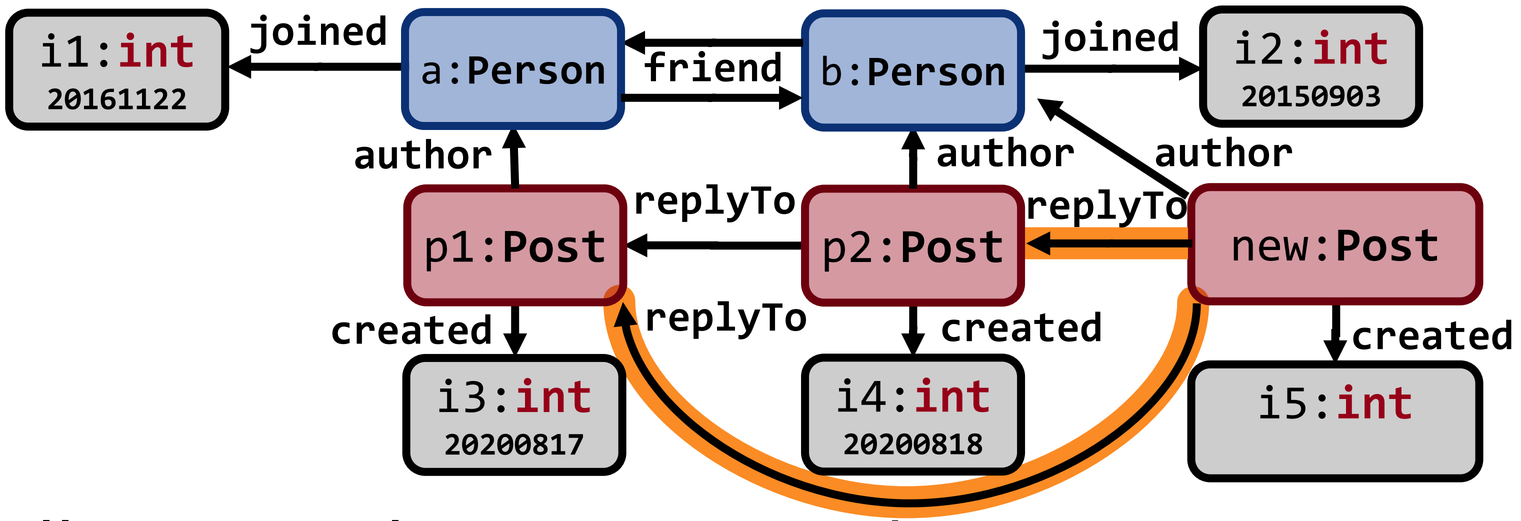
- Represent all potential extension with uncertainty
- Logic abstraction: true | false | unknown | error

Partial Modeling: 4-valued logic



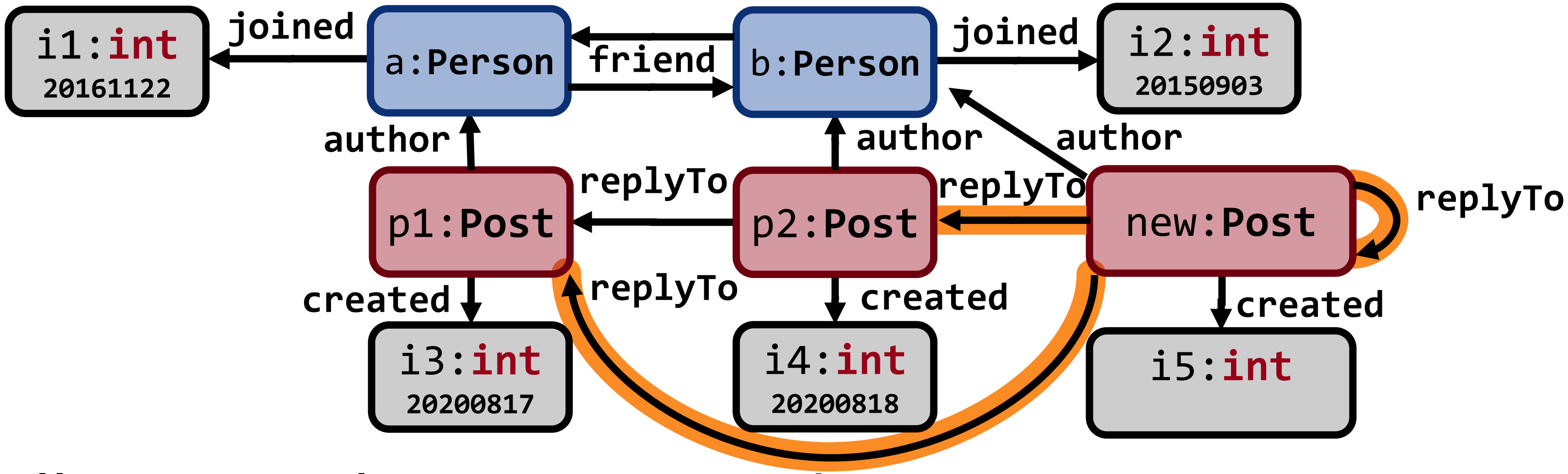
- Represent all potential extension with uncertainty
- Logic abstraction: true | false | unknown | error

Partial Modeling: 4-valued logic



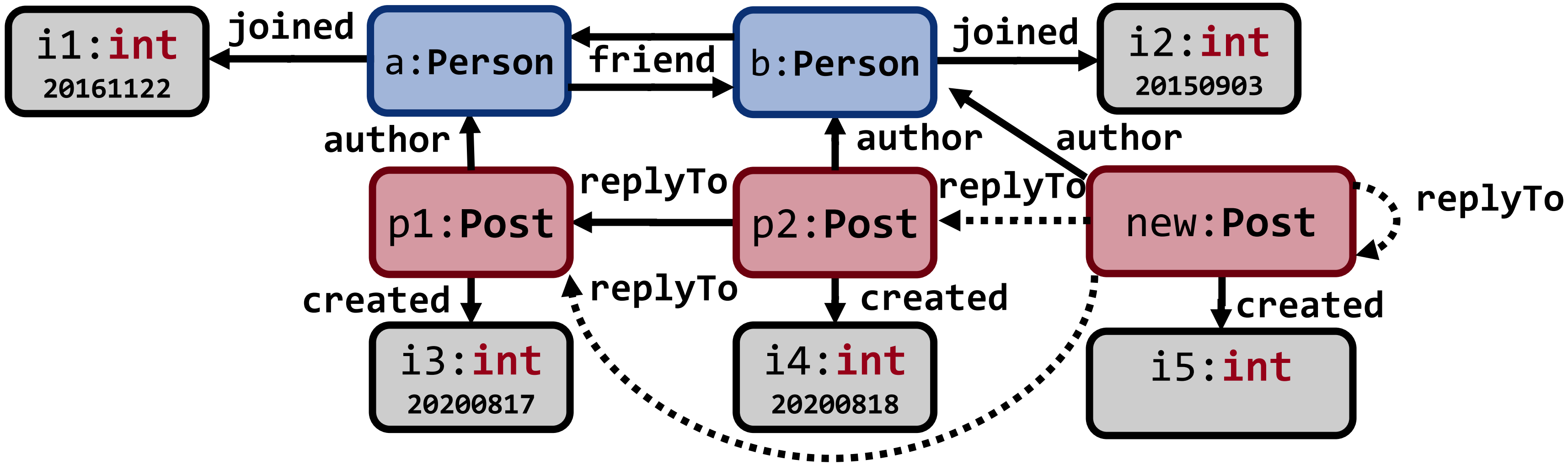
- Represent all potential extension with uncertainty
- Logic abstraction: true | false | unknown | error

Partial Modeling: 4-valued logic



- Represent all potential extension with uncertainty
- Logic abstraction: true | false | unknown | error

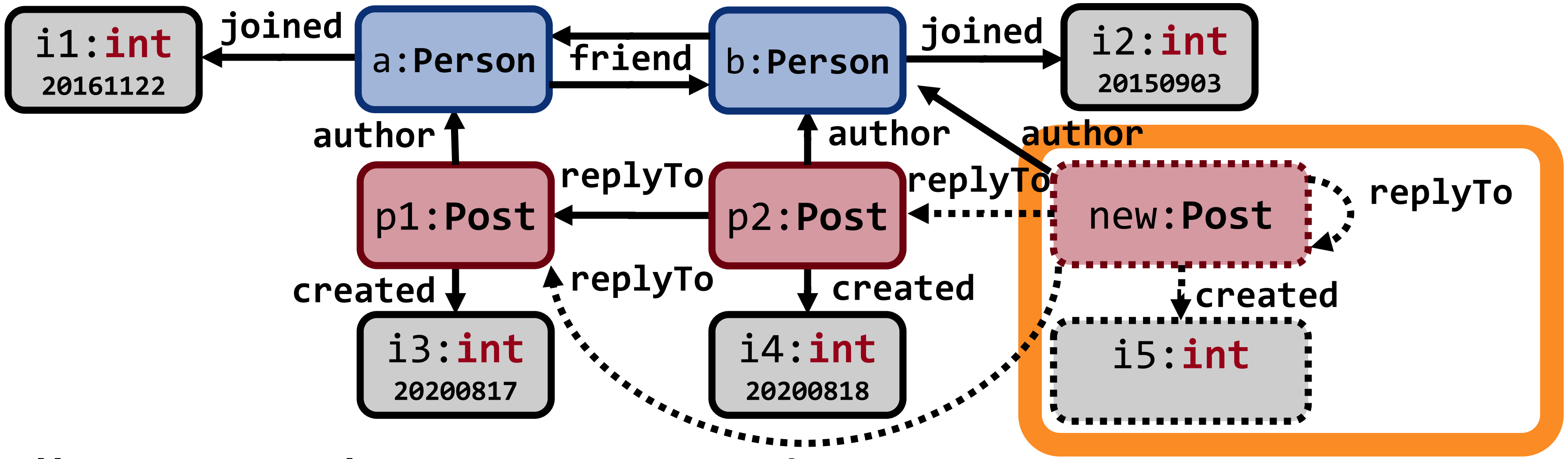
Partial Modeling: 4-valued logic



- Represent all potential extension with uncertainty
- Logic abstraction: true | false | unknown | error

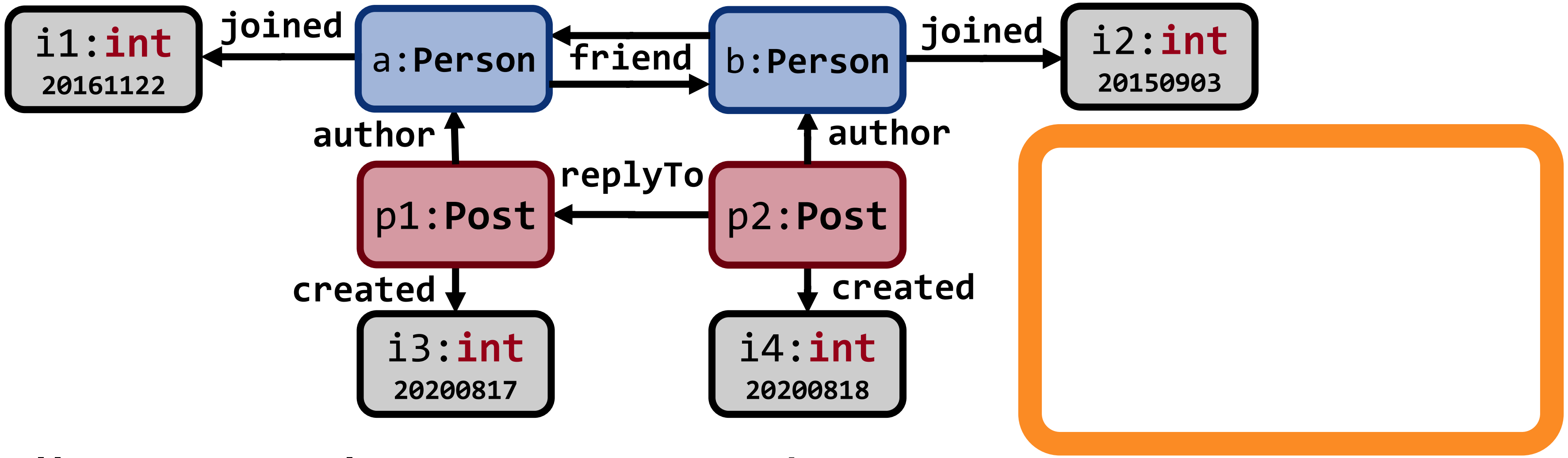
ftsrg

Partial Modeling: existence



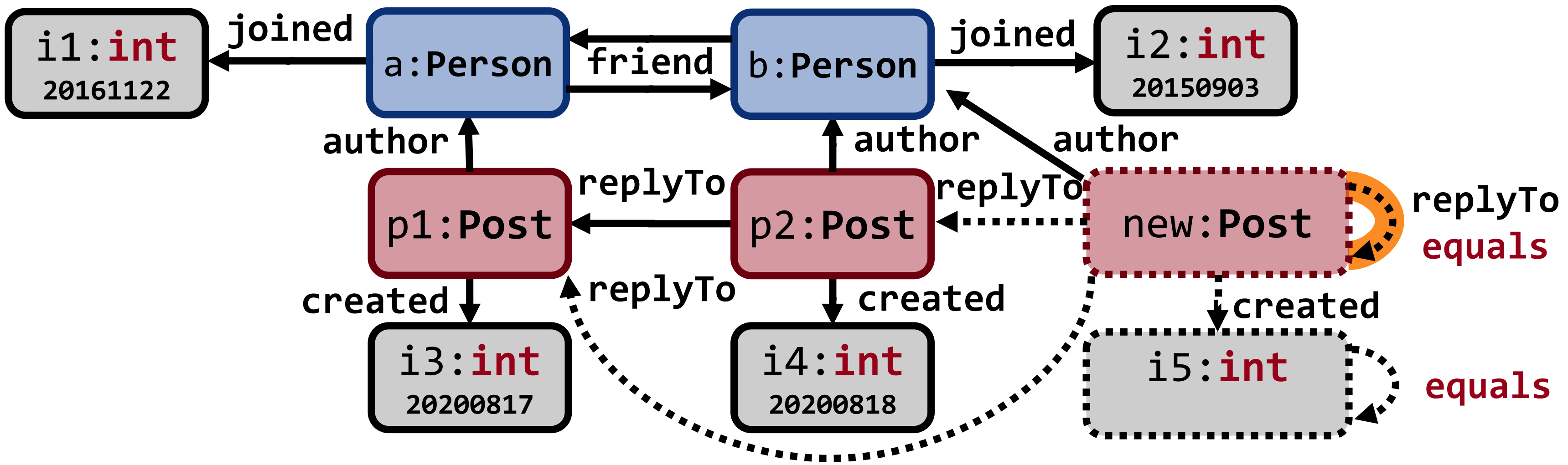
- Represent all potential extension with uncertainty
- Logic abstraction: true | false | unknown | error
  - 4-valued **exists**: added or removed

Partial Modeling: existence



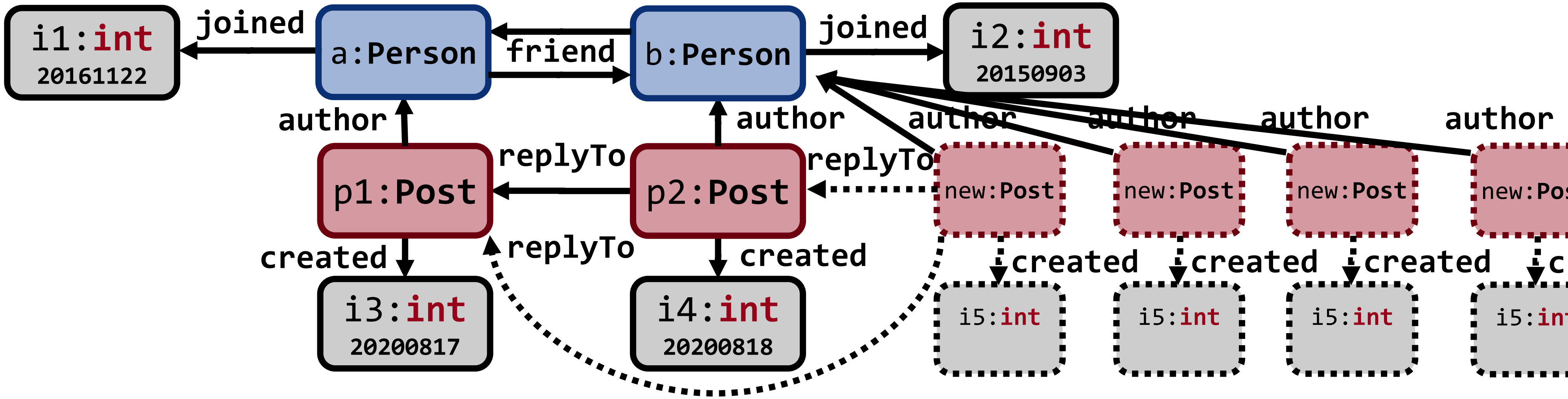
- Represent all potential extension with uncertainty
- Logic abstraction: true | false | unknown | error
  - 4-valued **exists**: added or removed

Partial Modeling: equivalence



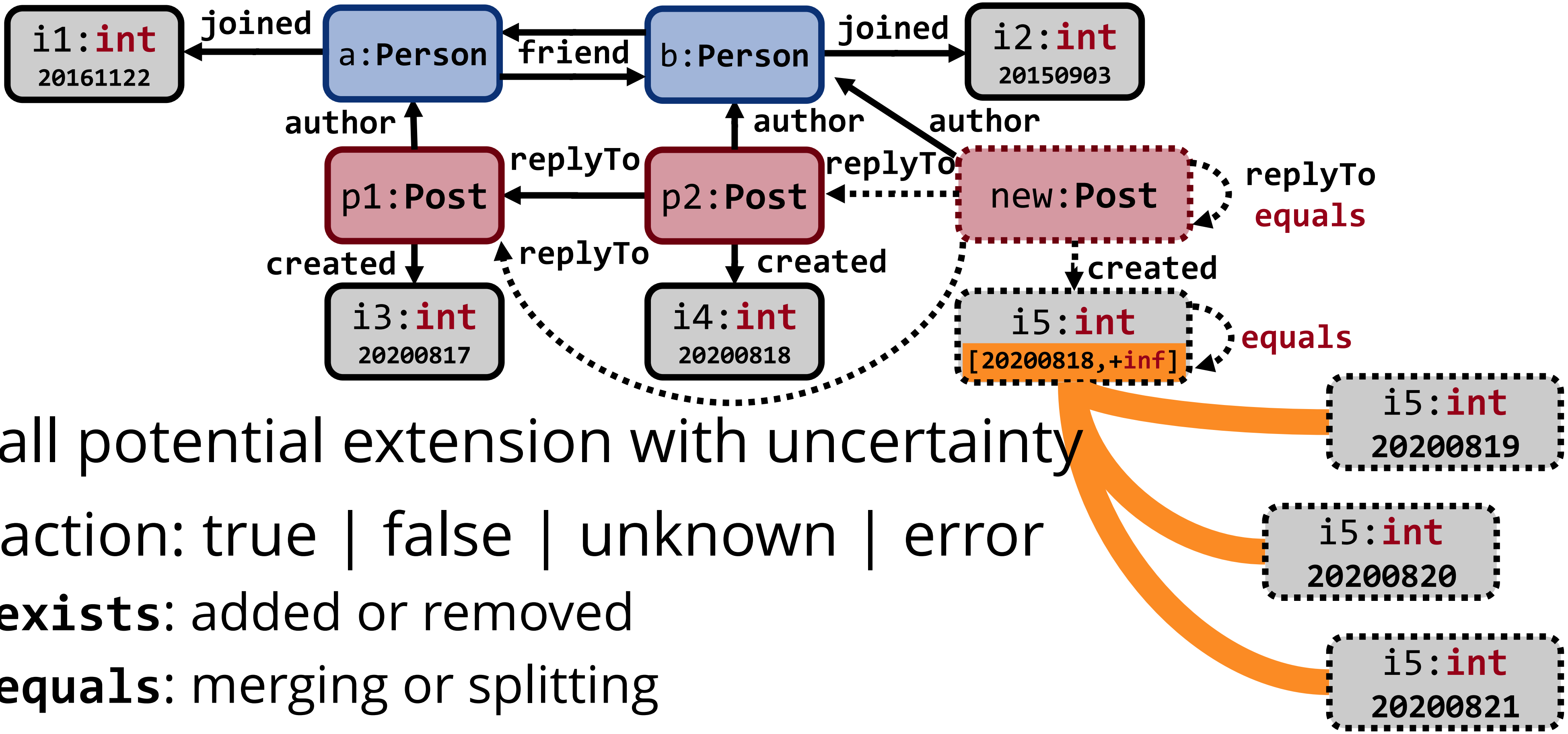
- Represent all potential extension with uncertainty
- Logic abstraction: true | false | unknown | error
  - 4-valued **exists**: added or removed
  - 4-valued **equals**: merging or splitting

Partial Modeling: equivalence



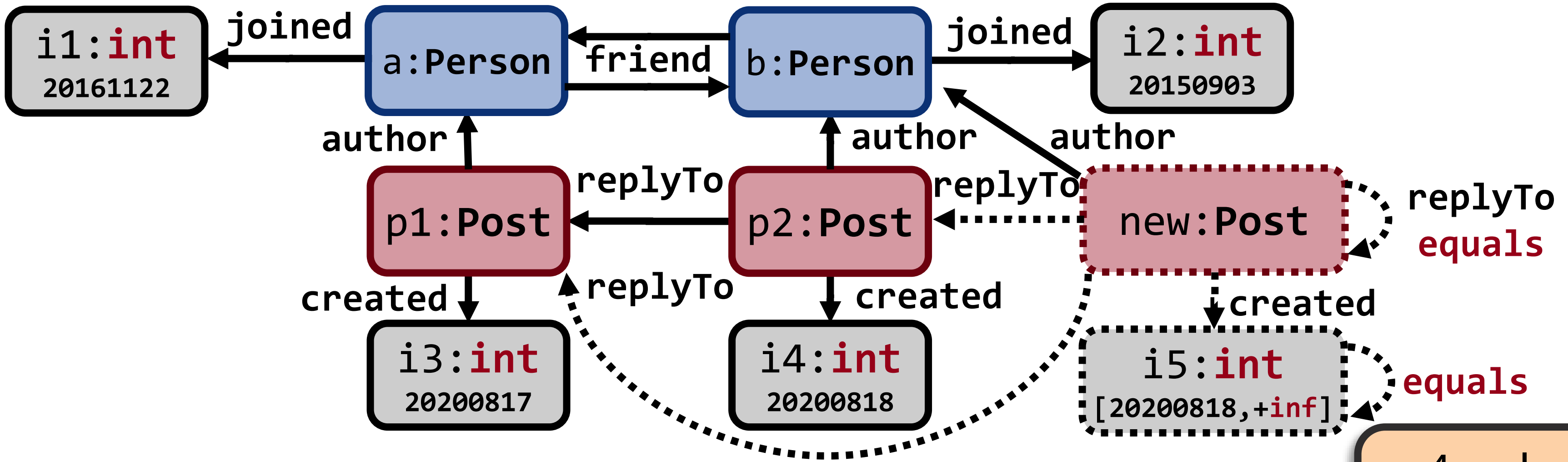
- Represent all potential extension with uncertainty
- Logic abstraction: true | false | unknown | error
  - 4-valued **exists**: added or removed
  - 4-valued **equals**: merging or splitting

Partial Modeling: numbers

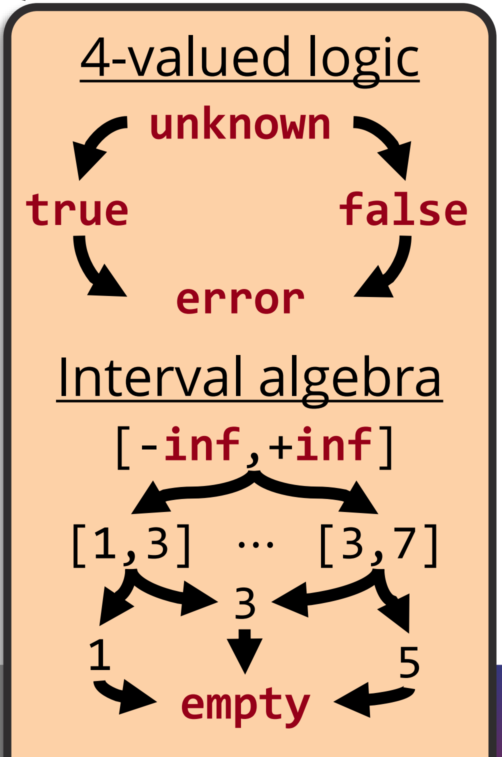


- Represent all potential extension with uncertainty
- Logic abstraction: true | false | unknown | error
  - 4-valued **exists**: added or removed
  - 4-valued **equals**: merging or splitting
- Numeric abstraction: concrete values → intervals

Partial Modeling: refinement



- Represent all potential extension with uncertainty
- Logic abstraction: true | false | unknown | error
  - 4-valued **exists**: added or removed
  - 4-valued **equals**: merging or splitting
- Numeric abstraction: concrete values → intervals
- **Refinement**: reduces uncertainty → concrete models



Refinement

- A **refinement** from partial model P to Q is defined by a refinement function $ref\colon O_P \rightarrow 2^{O_Q}$, which respects information ordering:
  - For all symbol $s \in \Sigma$: $I_P(s)(\bar{p}) \sqsubseteq I_Q(s)\left(\widehat{ref(\bar{p})}\right)$
  - All objects in Q are refined from an object in P, and existing objects $p \in O_P$ must have a non-empty refinement.

- A **concretization** is a refinement to a concrete model.

- **Regular models:** subset of partial models under analysis (e.g. exclude object merge, if impractical)

Thank you for your attention