Part I: Graph Transformation (8 points)

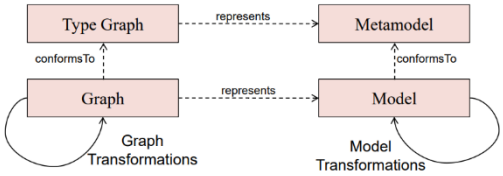
Briefly explain the concept of graph transformations and outline the structure of graph transformation rules based on an example that uses constants, variables, and expressions.

Graph transformations are a fundamental concept in model-driven engineering, allowing for the modification of graph-based structures. The structure of graph transformation rules can be understood through an example involving constants, variables, and expressions.

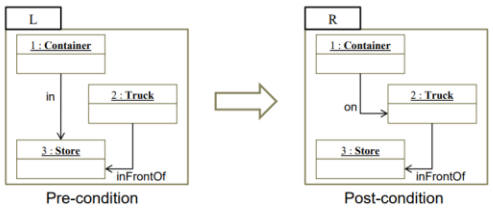
Consider a transportation model with containers, trucks, and stores. A graph transformation rule could describe the process of loading a container onto a truck. In this rule:

- **Models** are **graphs**
  - Class diagram, Petri net, state machine, …
- **Type Graph**:
  - Generalization of graph elements
- **Graph transformations**
  - Generalization of the **evolution** of graphs

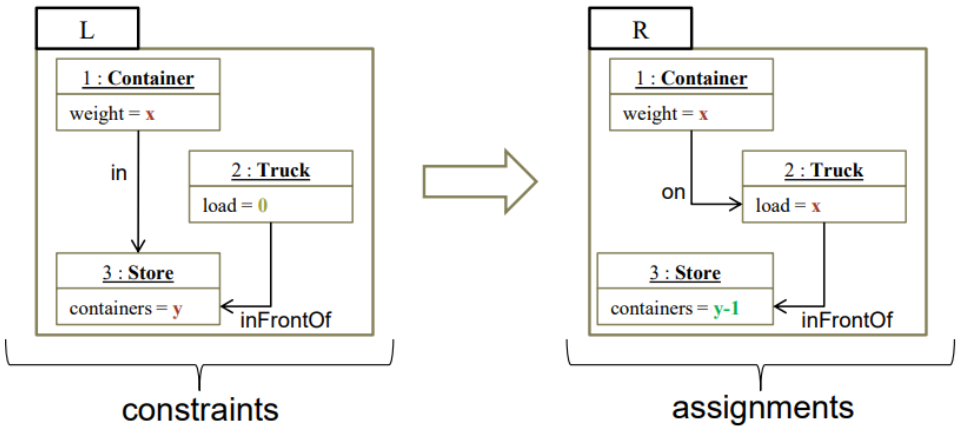
⇨ **Graph transformations are applicable for models!**



- A graph transformation rule *p: L → R is* a structure preserving, partial mapping between two graphs
  - L and R are two directed, typed and attributed graphs themselves
  - Structure preserving, because vertices, edges and values may be preserved
  - Partial, because vertices and edges may be added/deleted
- Example: Loading of Container onto a Truck



Constants
Variables
Expressions (OCL & Co)



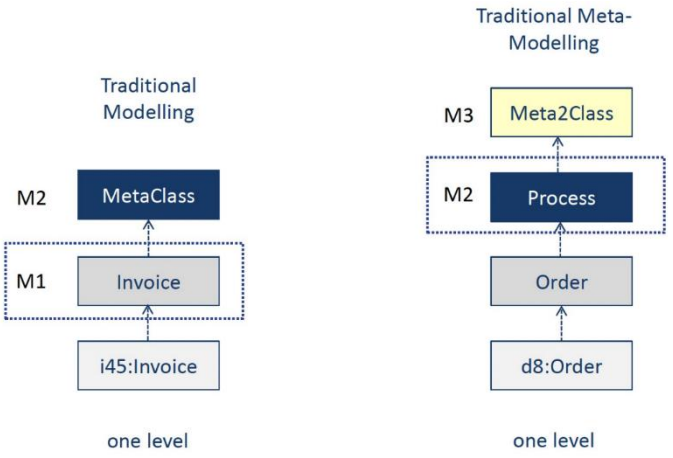
Part II: Multi-Level Modeling (8 points)

What is the unique characteristic of Multi-Level modeling compared to the OMG four level architecture? Define a minimalist example and use it to explain how language architectures are defined in a multi-level sense, and how this is or is not possible in the conventional four level architecture.

The unique characteristic of Multi-Level Modeling compared to the OMG four-level architecture lies in its ability to allow for an arbitrary number of classification levels. This is a significant departure from the traditional four-level architecture, which rigidly confines to only four levels (M0-M3).

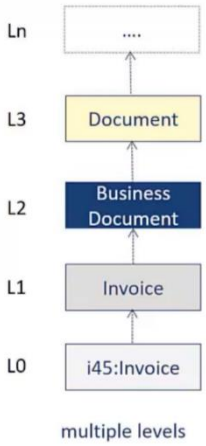
In a minimalist example, consider a scenario where we model a university. In a conventional four-level architecture, we would have levels like M0 (actual data, e.g., a specific course), M1 (models, e.g., a course template), M2 (metamodels, e.g., definitions of what constitutes a course), and M3 (meta-metamodel, e.g., language definition rules). However, this structure limits the ability to express more nuanced layers, like differentiating between undergraduate and postgraduate course templates, which could be another level in-between M1 and M2.

In Multi-Level Modeling, such additional levels can be seamlessly integrated, allowing for more expressive and detailed modeling that accurately represents the real-world complexity. This flexibility is not achievable in the conventional four-level architecture due to its fixed number of layers.



Observations/Limitations of the Traditional Approach

1. Lack of Expressiveness: *Instantiation* vs. *Specialization*
2. Distinction between *Language* and *Language Application*
3. Fundamental Design Conflicts
4. *Synchronization* between *models* and *code*



The Promise of Multi-Level Modeling

- ❑ new language paradigm
- ❑ allows for an arbitrary number of classification levels
- ❑ motivated by the lack of abstraction in traditional, MOF-like language architectures
- ❑ first introduced in 2001 by Atkinson and Kühne
- ❑ with roots going back to the early 90s
- ❑ various approaches developed since then
- ❑ focus mainly on modeling, not on programming languages

Part III: Web Modeling (6 points)

Briefly describe the scope of the language server protocol in general and further describe the role it plays in the Eclipse Graphical Language Server Protocol Platform (GLSP). Describe two needs for flexibility that are supported by GLSP-based web modeling tools.

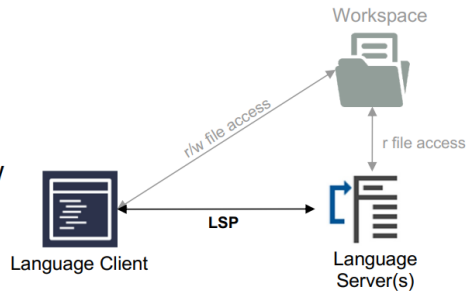
The Language Server Protocol (LSP) is designed to standardize the way tools and servers communicate for programming languages. It essentially decouples language-specific features from the editor, enabling the editor to support any language. In the context of the Eclipse Graphical Language Server Protocol Platform (GLSP), the LSP's principles are applied to graphical modeling. GLSP facilitates the development of browser-based diagram clients, focusing the frontend on rendering and user interaction while the backend encapsulates the language specifics.

Two key needs for flexibility supported by GLSP-based web modeling tools include:

- Customizability: GLSP allows for the creation of domain-specific functionalities and customizing default behaviors. This is crucial for adapting to different domains, workflows, and integration with other tool components.
- Rendering and Editing Flexibility: GLSP supports custom rendering of diagrams and provides extensible editing tools. This is important for creating graphical representations specific to different domains and for providing tailored editing experiences.

Language Servers: The Basic Idea

- Encapsulate language implementation in a server
  - Parsing, AST, indexing, validation, etc.
  - Re-use existing language support
- Generic text editor client
  - Focus on user interface
  - Generic text editing functionality
- Protocol that transfers language know-how
  - From server to client
  - On a need-to-know basis
  - In a UI-oriented way



Eclipse Graphical Language Server Platform

Applying the architectural pattern of LSP to graphical modeling

- Development of browser-based diagram clients
- Frontend focused on rendering & user interaction
- Encapsulate language smarts on the diagram server

True/False questions

[X] In ATL, lazy rules are applied once for each match found in the input model.

[✓] In Henshin, negative application conditions are defined with the forbid action.

[✓] SysML v2 comes with a textual concrete syntax.

[✓] "Do" blocks of ATL rules are inherited to subrules.

[✓] The left side of a graph transformation rule specifies what must be existing in a concrete graph to execute the rule.

[X] Meta-markers of template-based model-to-text transformation languages are used to define static text blocks.

[X] In-place model transformations build a new model from scratch.

[X] Xtend dispatch methods are required to invoke the code generation process.

[✓] Mode-to-model transformations are used to automatically create target models from source models.

[X]Model-to-model transformations always have a single model as input and a single model as output.

[✓] To change a single attribute value an in-place transformation is usually more efficient compared to an out-place transformation

[✓] Transformations need to conform to a transformation metamodel to be applicable

[X] A transformation execution engine takes a transformation model and a source model as input and produces a target metamodel

[ ] Xtend enables code generation with (M2M, M2T, programming languages,..) and is a dialect of (Java, HTML, Henshin,..)

[X] Negative application conditions are applied in cases where no other rules match.

[✓] Rules as well as units can have input parameters in Henshin.

[✓] A Henshin rule is only applief if for each preserve node a matching element exists in the source model.

[✓] The target model of a Henshin rule conforms to the same metamodel as the source model.

[X] So far, LSP is only used by Eclipse and VSCode

[X] The EMF distribution used in this course is a good example for an application enabled by LSP.

[X] Language servers usually run on the users machine, in order to provide fast language support in the editor.

[✓] LSP and language servers would enable us to implement an online modeling tool for our textual SBSML from lab2.

[✓] LSP is used by many different language servers, among others supportin Java, Haskell and Typescript

[X] In ATL source models can be read and written

[X] Matched rules and lazy rules are only applied once for each element

[✓] Matched rules have no side-effects

[✓] Helper functions can have an optional context