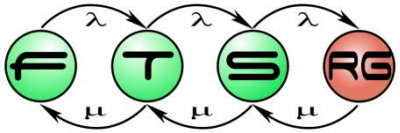
Graphical Concrete Syntax Design for Domain-specific Languages

Model Driven Software Development

Lecture 6

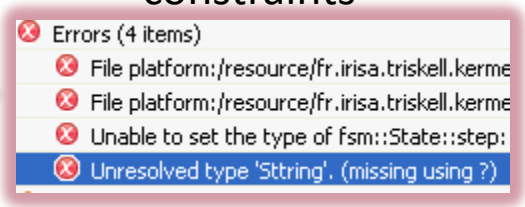


Structure of DSMs

Graphical syntax

Abstract syntax

Well-formedness constraints

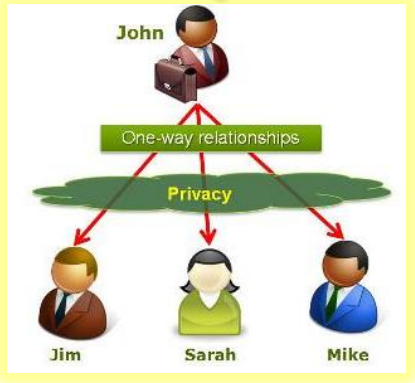


Behavioural semantics, simulation

Code generation

Mapping

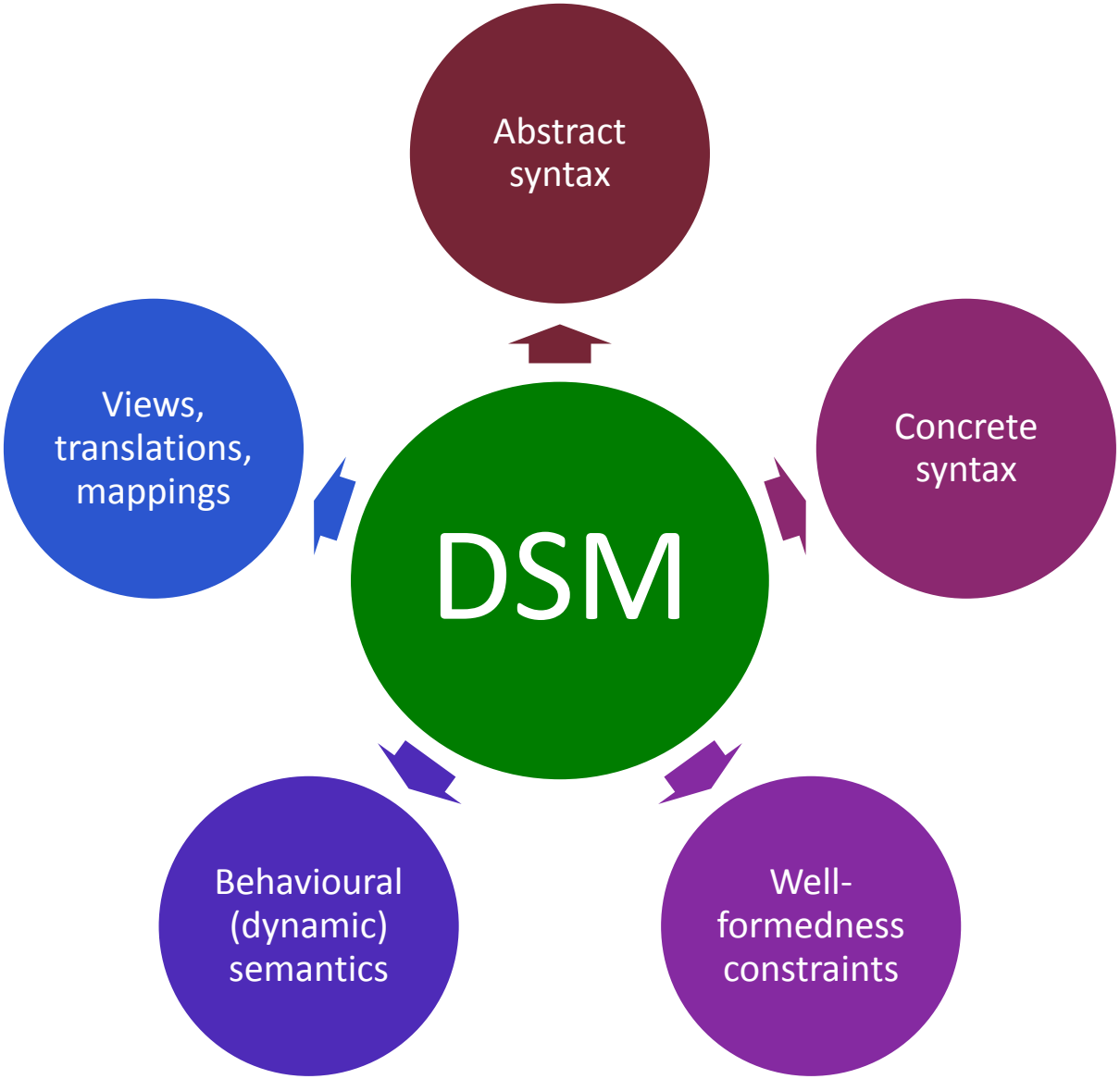
Textual syntax



View

Code (documentation, configuration)

DSM aspects


Abstract syntax
Views, translations, mappings
DSM
Concrete syntax
Behavioural (dynamic) semantics
Well-formedness constraints

Concrete Syntax Design

- User-managed parts of a modeling language
  - Performance
  - Robustness
  - Usability issues
- Creating model editors
  - Similar problems at programming languages
  - IDE extensions needed

Approaches

- Textual syntax
  - Character-based edit operations (unless *projectional*)
  - Abstract syntax: traditional AST
- Graphical syntax
  - Editing operations: translated to abstract syntax
  - Abstract syntax: based on metamodel
- Form-based entry
  - Less common
  - Behaves similar to graphical syntax

Advanced features

High level editing support

- Outline view
- Documentation display (e.g. Javadoc)
- Templates/snippets/examples
- Content assist
- Validation, automatic fixes

Project-level integration

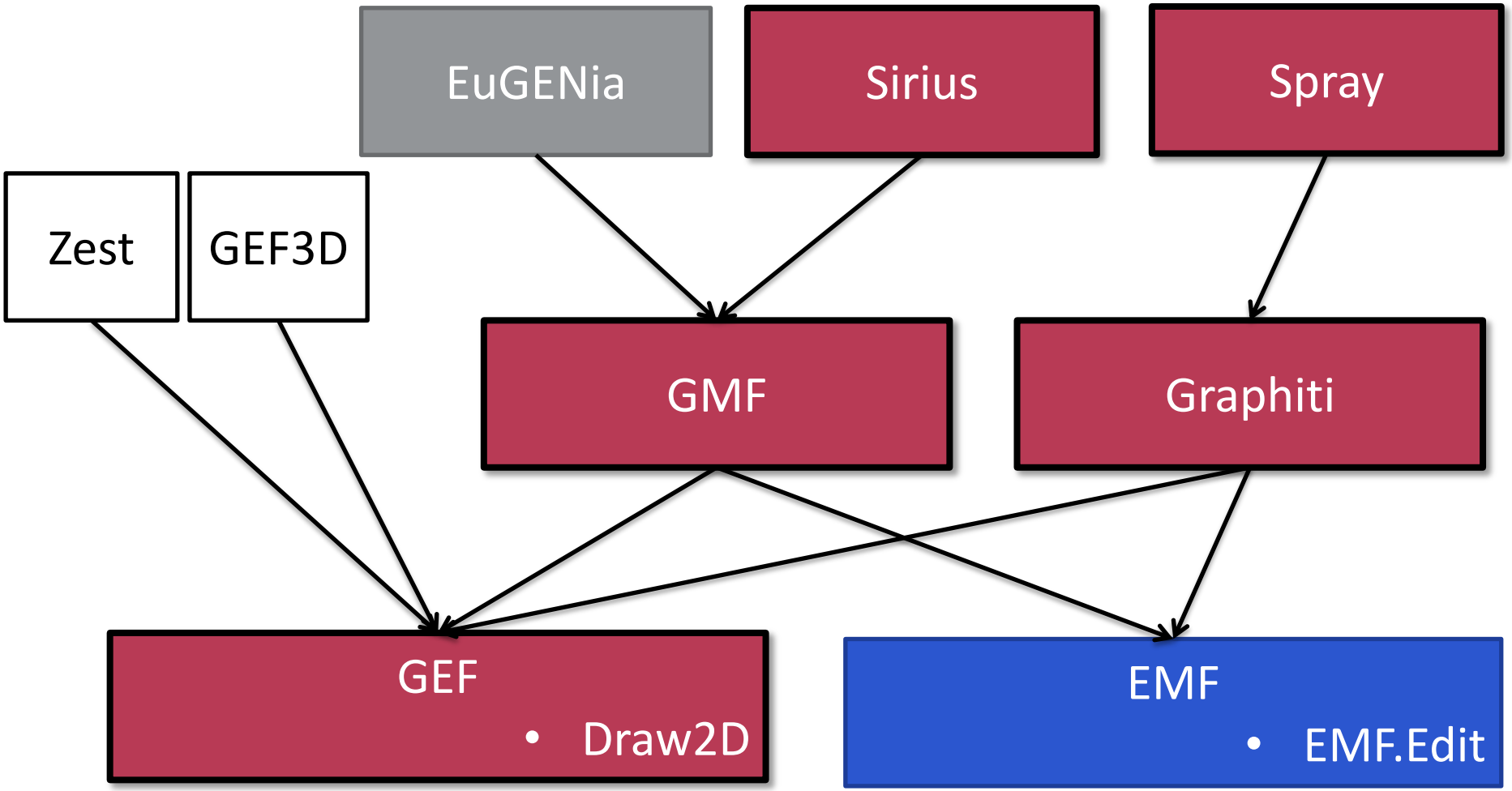
- Code generation
- Wizards to create projects/files
- Integration with manually written code in programming language

Technology

- Eclipse Modeling Tools
  - Several related subprojects
  - Each supports a single aspect
  - Examples of today
- Microsoft Visual Studio 2010 Visualization & Modeling SDK
  - DSL modeling framework from Microsoft
  - Own metamodeling core
  - Focuses on graphical modeling

Graphical Editors

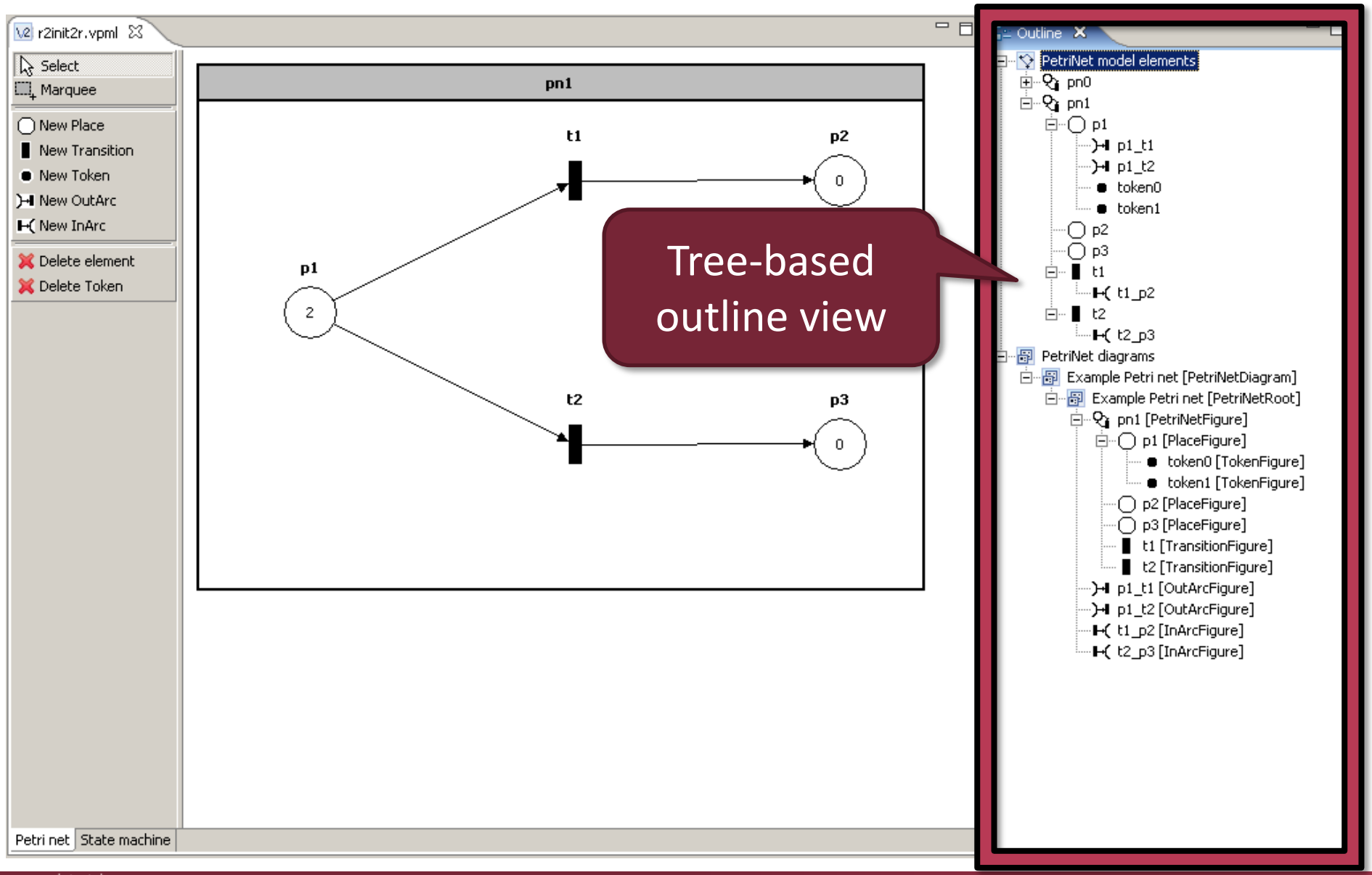
Graphical Editor Techiques



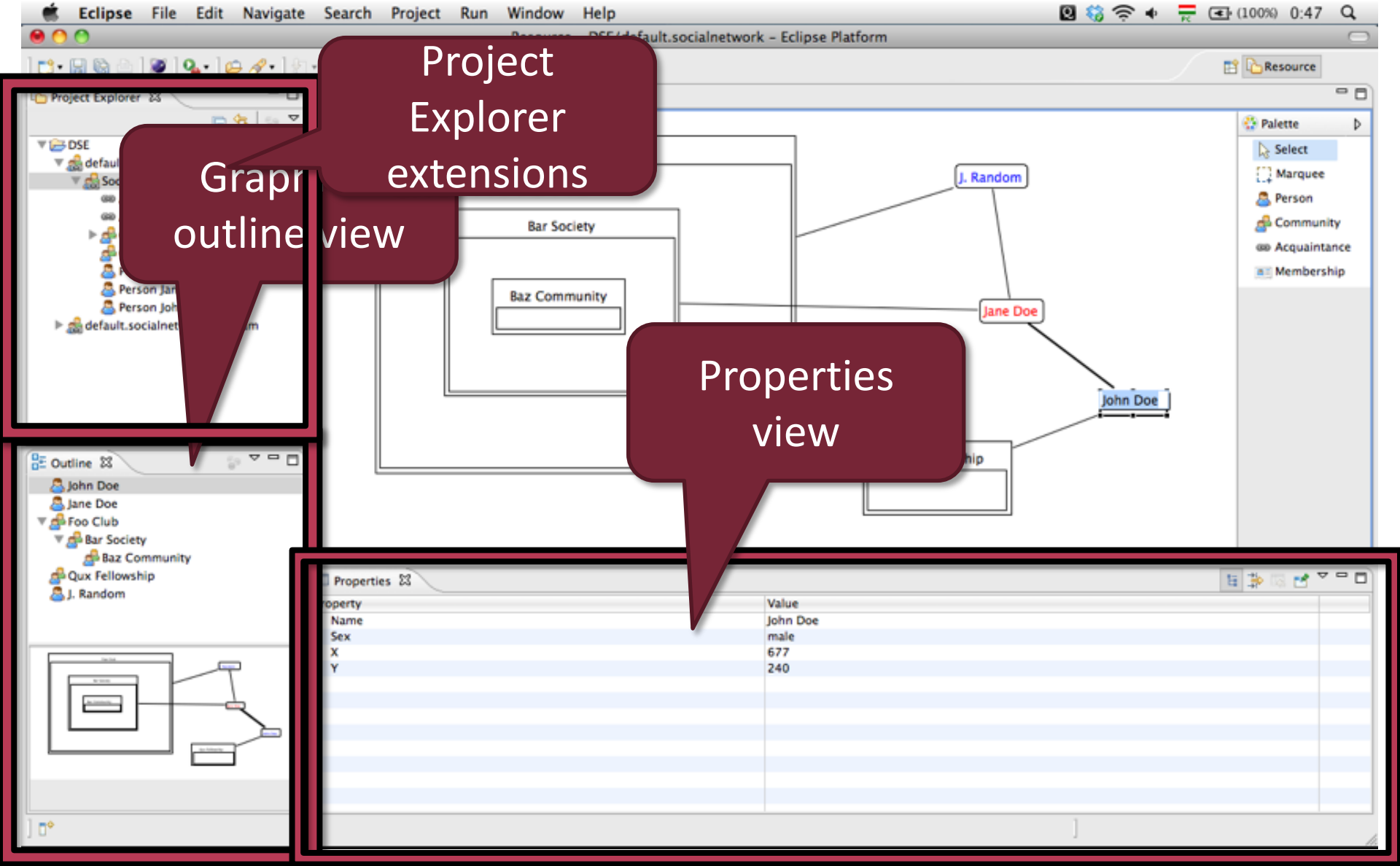
Graphical Modeling

- Model
  - Typically graph-based modeling (Edges, Nodes)
  - In our case EMF
- Idea
  - Display and editing as a graph model

Example: Petri net editor



Example: Social Network editor



Implementation

- Presentation
  - Based on a Canvas
  - Using vector-graphic libraries (GEF/Draw2d)
- Model manipulation
  - EMF Edit model manipulation commands
    - Atomic operations: create/modify/remove node/edge
  - Transactional modifications
    - Undo/redo support
    - Replayability

Implementation 2.

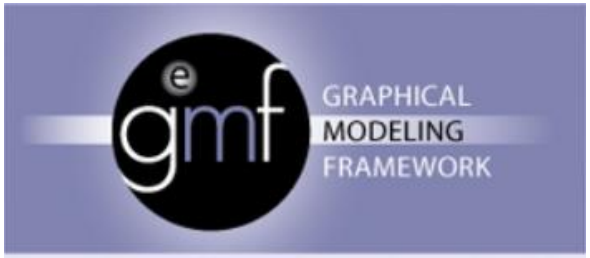
- View models
  - Modeling for view-specific information
    - Coordinates
    - Size
    - Colors and fonts
    - …
  - Generic implementation in GMF and Graphiti
  - Often stored in external files
    - Separation of concerns
    - E.g. code generator not interested in view information

Technologies 1. - GEF

- Graphical Editing Framework (GEF)
  - “Low level” editor framework
  - Not EMF-specific
- Model-View-Controller approach
- Generic graph-based editor framework
  - Including undo/redo support
  - Graphical outlines
- Manual coding for every possible element
- GEF4 FX – JavaFX-based replacement of the core

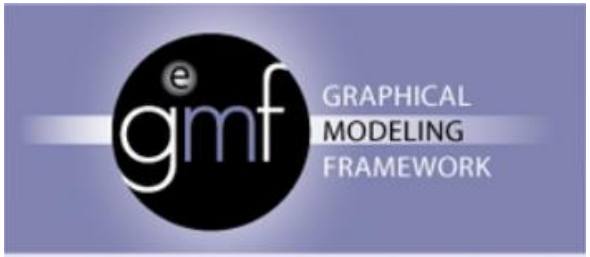
Technologies 2. – GMF

- Graphical Modeling Framework
- Based on GEF and EMF
- Well-separated view and domain models
  - Generic view model
  - Synchronization provided by GMF framework
- Relatively old technology
  - Widely used
  - Very complex to start



Technologies 2. – GMF

- Model-driven development environment
  - Common model for graphical editors, using
    - Figure definition model
      - Basic symbol definition of the graphical language
    - Tooling model
      - Defining model manipulation commands
    - Mapping model
      - Mapping figures and tools to domain model
  - Fully functional editor can be generated
    - Problematic manual modifications
- Or a high-level editor framework
  - Manual coding



Technologies 3. - Graphiti

- Newer high level graphical editor framework
  - Based on EMF and GEF
  - But: different approach then GMF
    - Simplified programmatic API
    - Manual coding
  - Idea
    - All Graphiti based editors should
      - Look similar
      - Behave similar

Technologies 3. - Graphiti

- Development methodology
  - Coding over a high-level Java framework
    - Much simpler then GMF
    - Repetitive code needed
- Spray project
  - Textual modeling environment for graphical editors
  - Generates code over the Graphiti framework

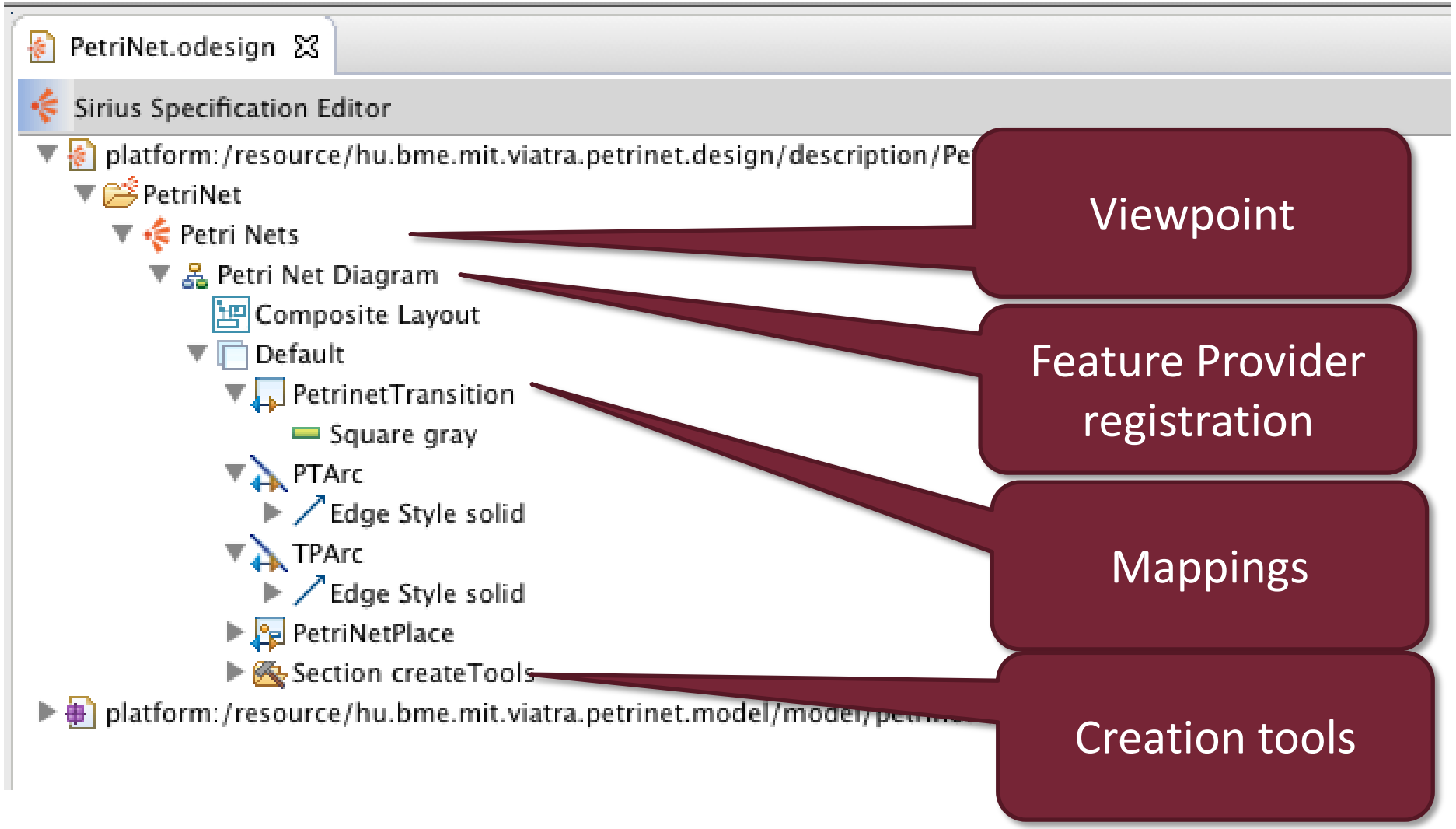
Technologies 4. - Sirius

- New modeling project
  - Since 2013 on eclipse.org
  - Previously Obeo Designer – commercial tool
- How stable is it?
  - Old projects are to be migrated
  - Version history
    - 0.9: 2013-12-10
    - 1.0: 2014-06-25 (Kepler release train)
    - …
    - 4.0: 2016-06-22
    - …

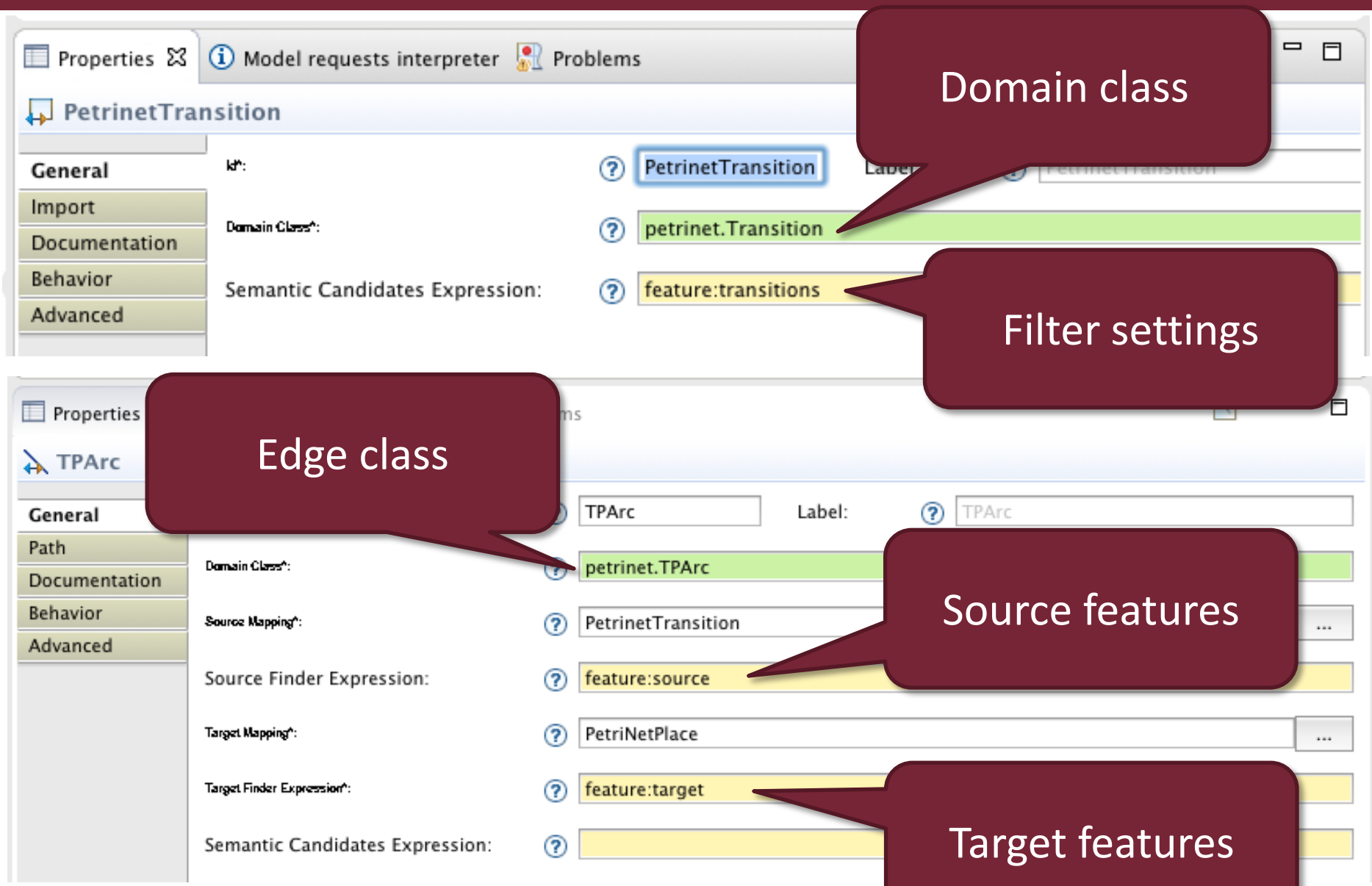
Sirius Viewpoints

- Base concept:
  - Every diagram is a view of the model
  - With a defined syntax
    - **Graphical**
    - Table/Tree syntax
    - Xtext-based textual syntax
- Viewpoint definition
  - Viewpoint specification model

Viewpoint Specification Model



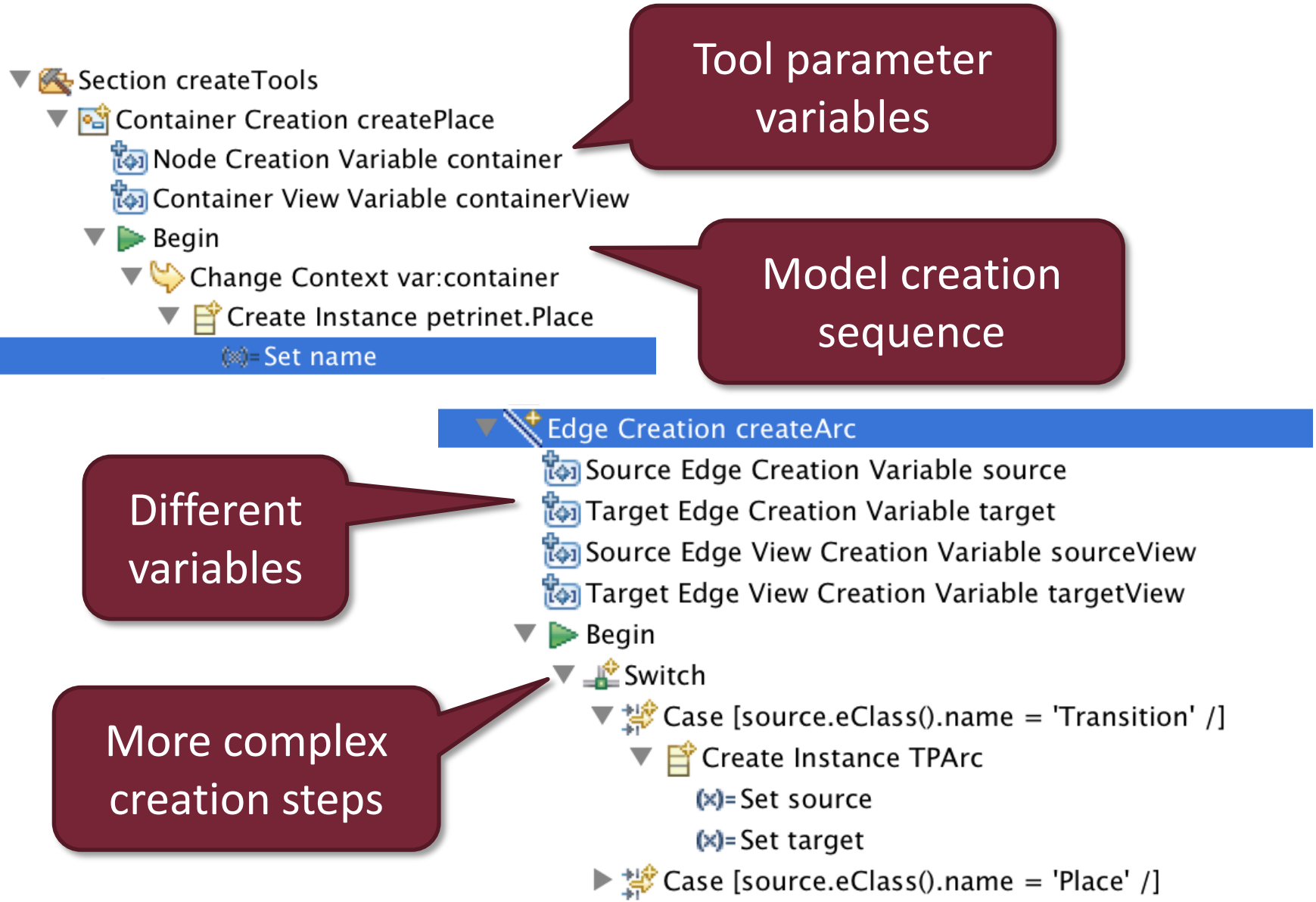
Node & Edge Mapping



Feature Selection

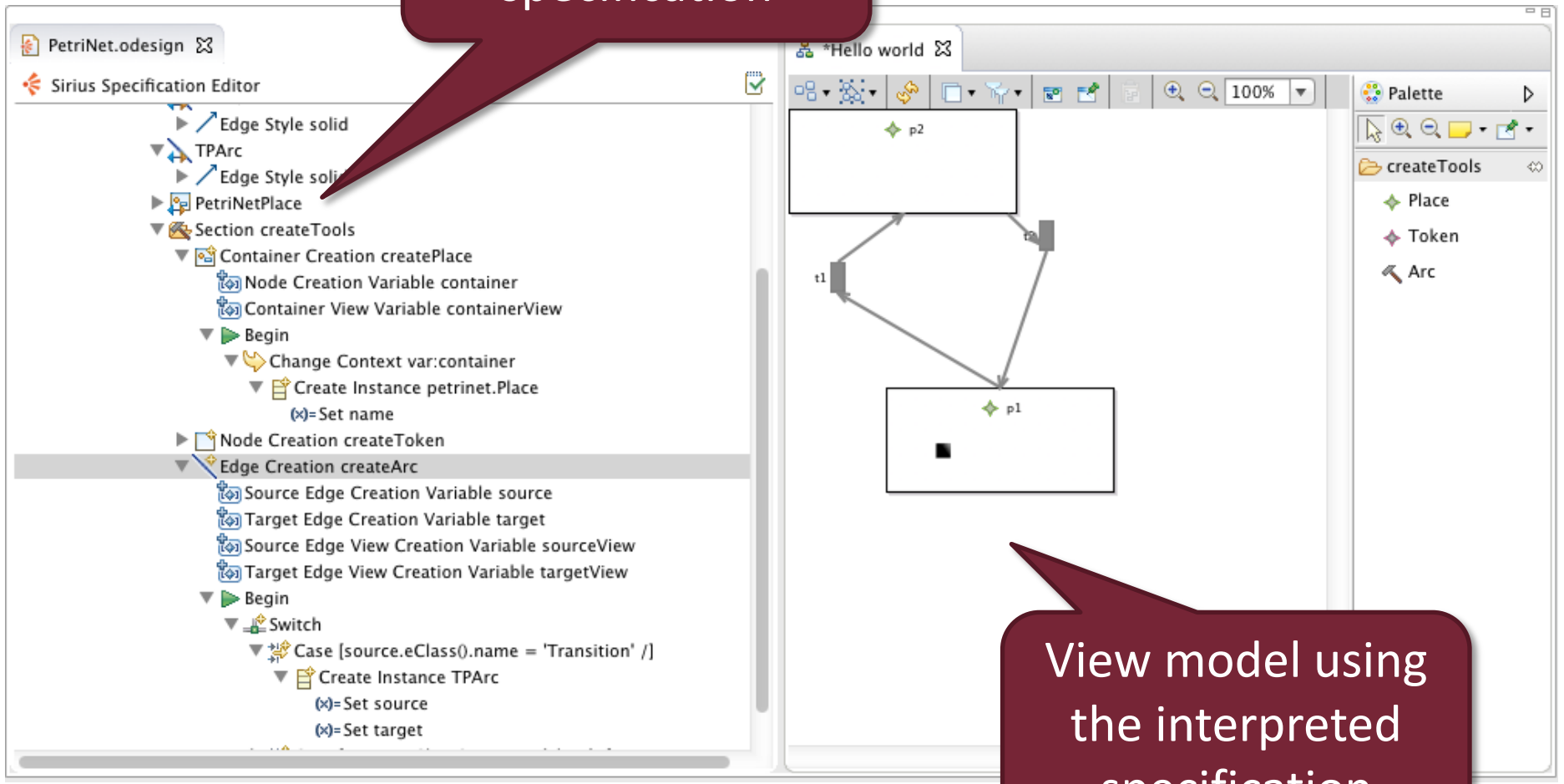
- Interpreted expressions
  - Special interpreters
    - **var**: accessing specification model variables
    - **feature**: accessing EMF model features
    - **service**: accessing service methods
  - Acceleo
    - Acceleo expressions
      - Basic operations
      - Comparison with single ‘=‘ symbols
    - Syntax: **[theExpression/]**
  - Raw OCL
    - Not recommended, Acceleo provides superset features
  - Custom interpreter

Node & Edge Tool



Interpreted Modeler Development

Viewpoint specification



View model using the interpreted specification

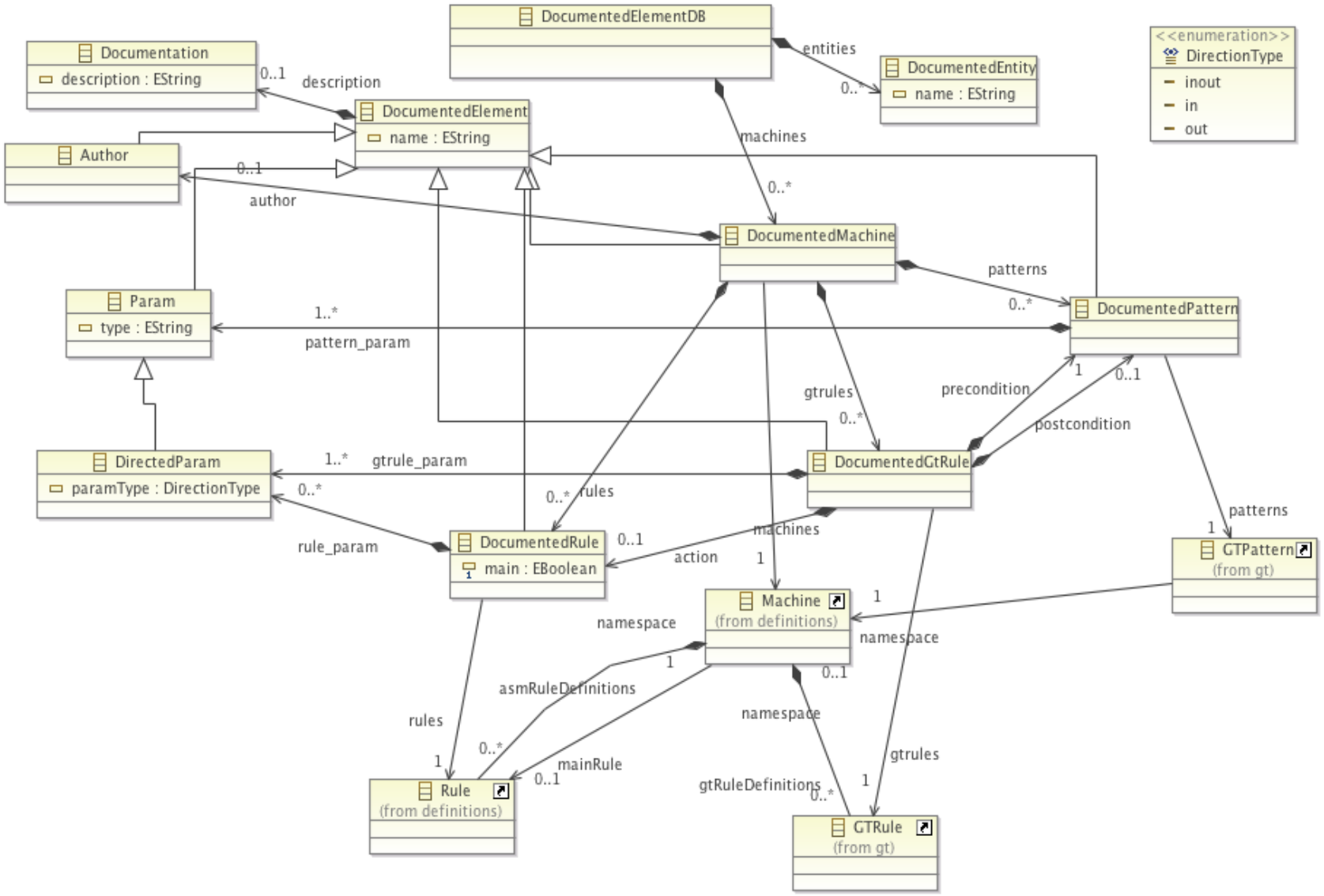
Technology Comparison

	GEF	GMF	Graphiti	Sirius
Model	Arbitrary	EMF	EMF	EMF
Non graph-based presentation	Manageable	Large amount of customization needed	Not supported	Tree, Table
Code size	Large, repetitive code	Mostly modeling, some coding	Smaller amount, but repetitive code	Negligible
Development workflow	Only coding	Modeling and coding	Coding	Modeling

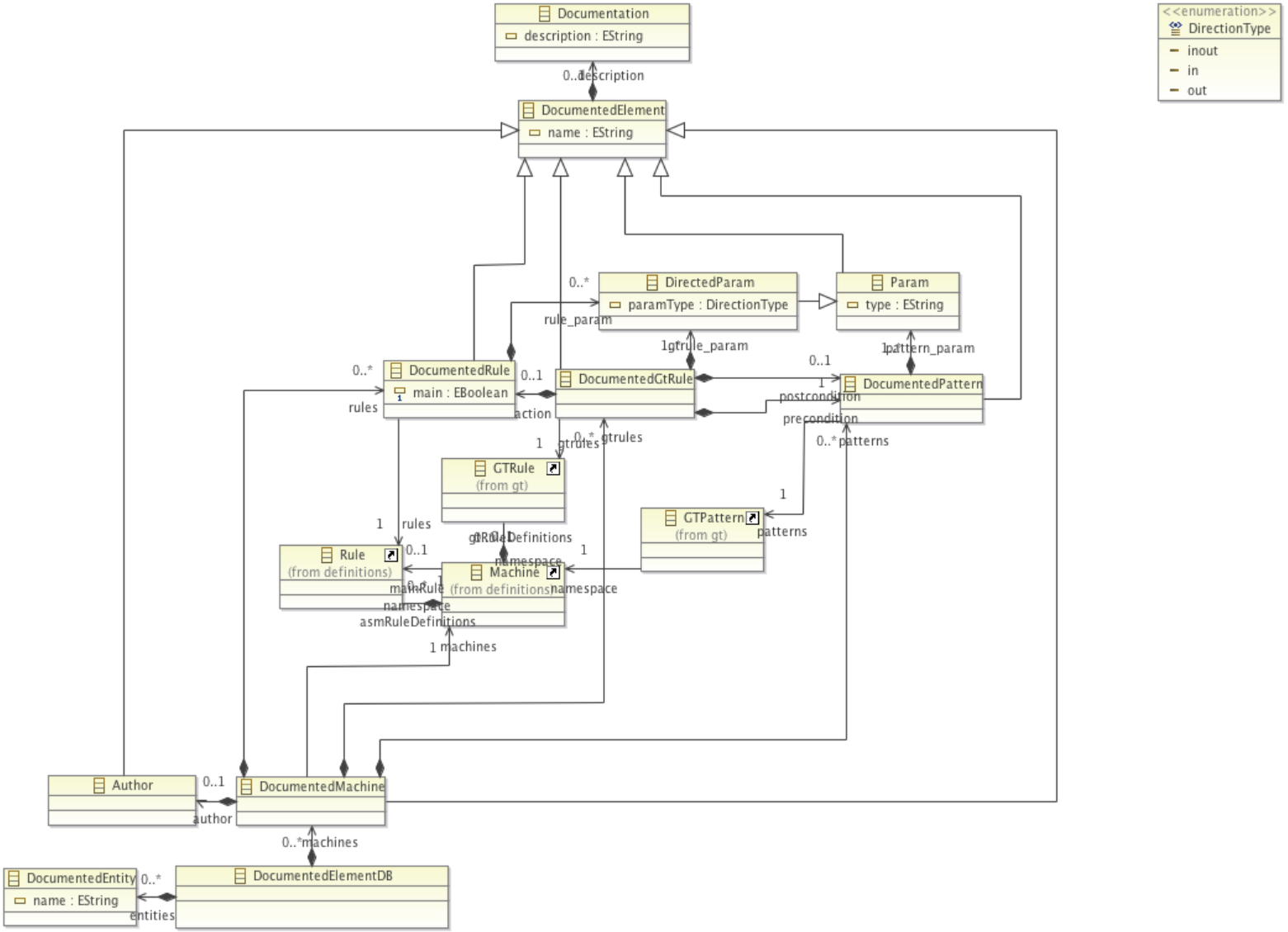
Advanced issues

- Cumbersome editing
  - E.g., reorganization to insert a node to the middle
- Handling large models
  - 20+ nodes on a diagram:
    - Logical structure, readability possible
    - But needs human support
  - 100-1000+ nodes on a diagram
    - Technological limitations
    - Usability limitations

Example: Layouting

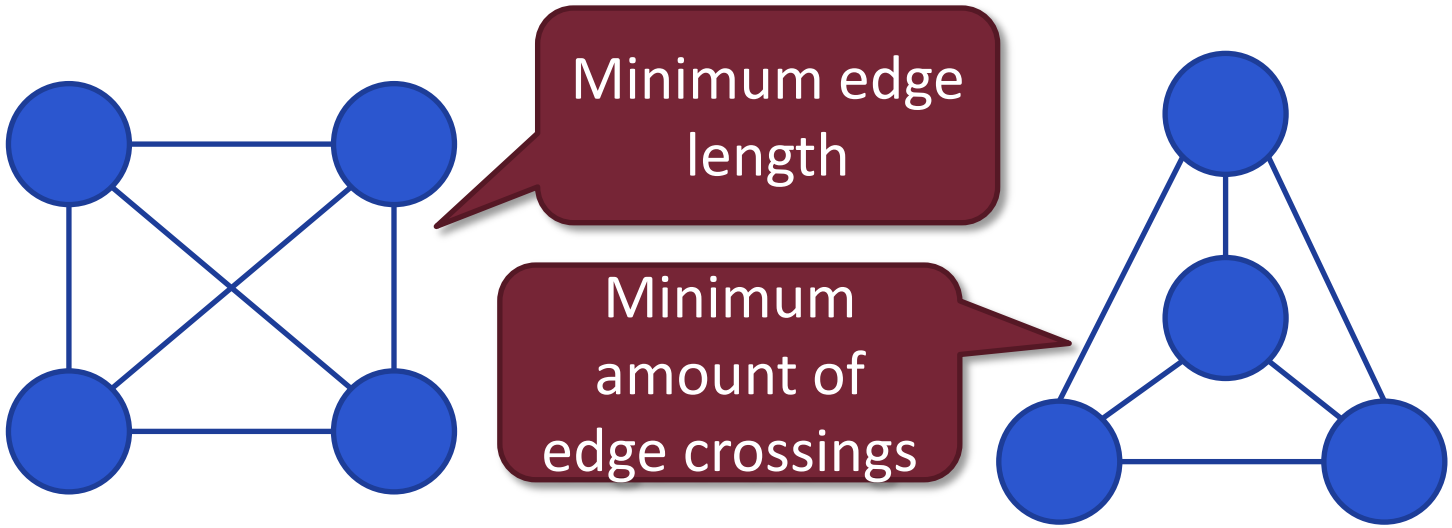


Example: Layouting



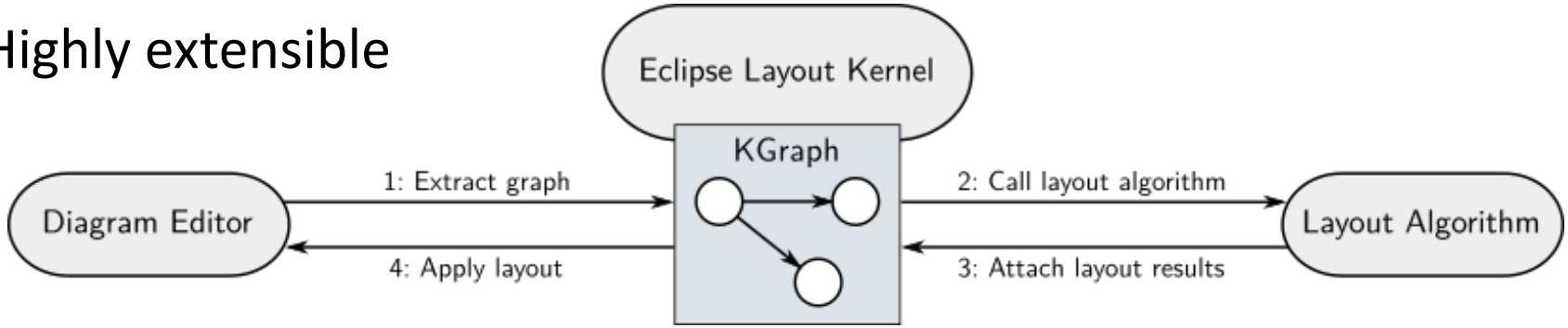
Layouting Support for Graphical Editors

- Computation of the position of nodes
  - Possible to do automatically
  - For a given metamodel
    - No unified visual requirements possible
    - We have to decide what is important to show



Layouting Support for Graphical Editors

- **GraphViz** - http://graphviz.org
  - Layouting project with high quality layout algorithm
  - Hard to integrate into Eclipse applications
- **Zest** - http://wiki.eclipse.org/index.php/Zest
  - Easily Eclipse integration (SWT-based graph widget)
  - So-so layout algorithms
- **ELK** (née ~~KIELER~~) - https://www.eclipse.org/elk/ (relatively new)
  - Eclipse Layout Kernel
  - Some built-in support: GMF, Graphiti
  - Highly extensible

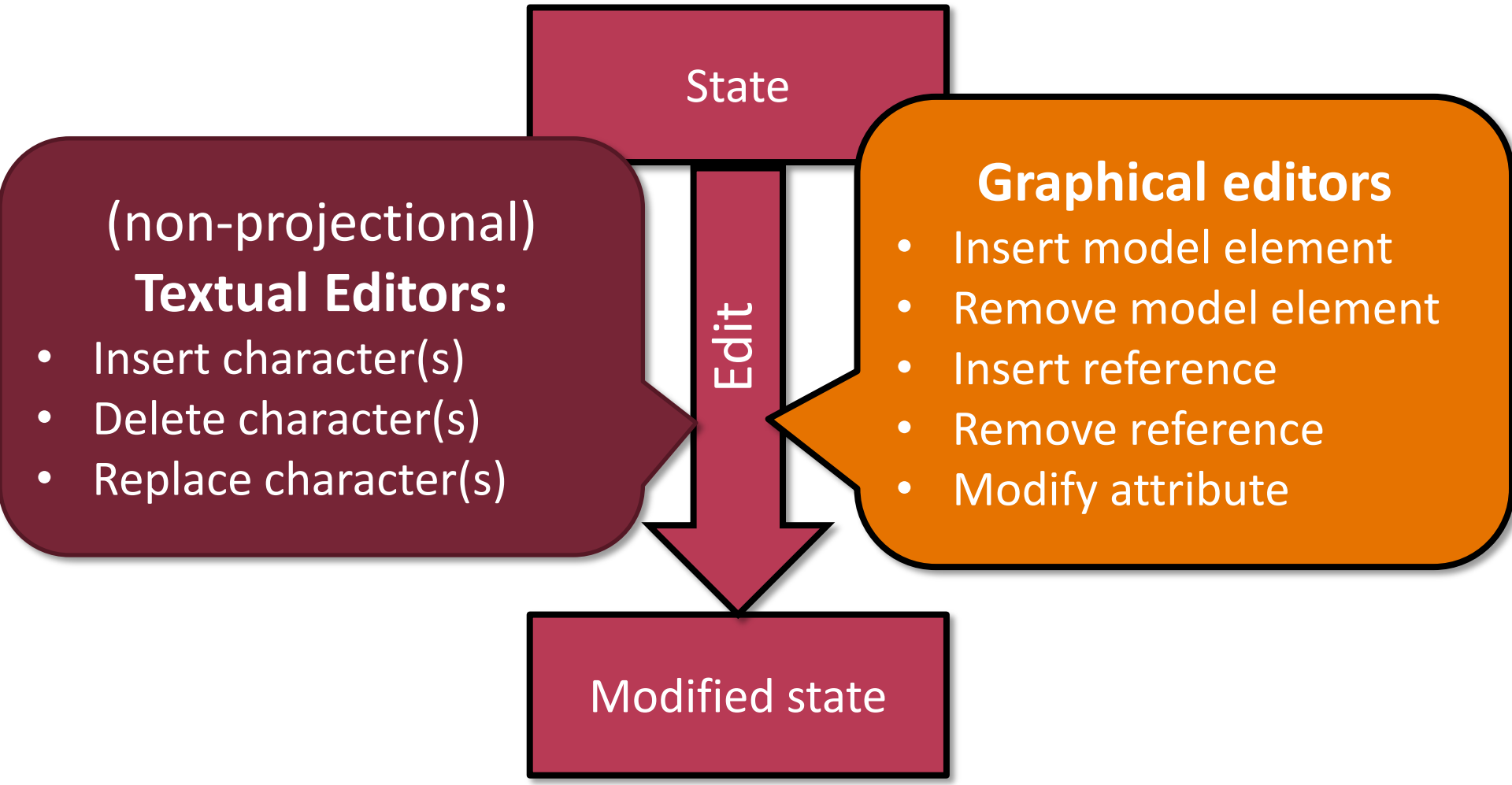


Textual or graphical?

Comparison

Textual Languages	Graphical Languages
Quick and simple editing	More cumbersome editing
References described as *string identifiers*	References displayed visually
Inconsistent models during editing	Models always syntactically correct
Automatic formatting	Automatic layouting
Content assist	Tool list to add nodes/edges
Displaying validation errors, offering quick fixes	
Both are supported with EMF-based technologies	

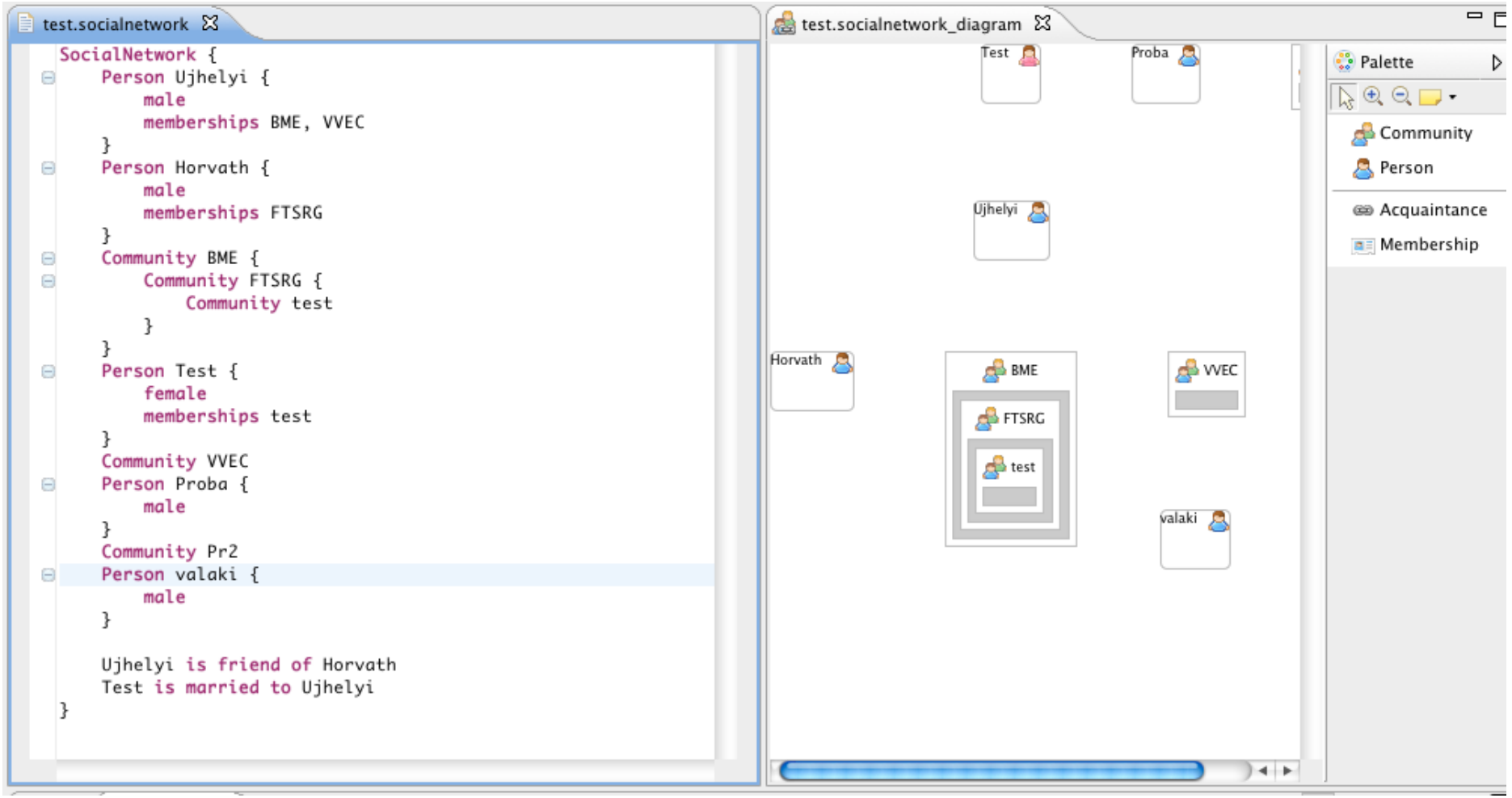
Editing


State
Edit
Modified state
(non-projectional)
Textual Editors:
• Insert character(s)
• Delete character(s)
• Replace character(s)
Graphical editors
• Insert model element
• Remove model element
• Insert reference
• Remove reference
• Modify attribute

Question: textual or graphical?

- No clear choice
- Rules of thumb
  - Behaviour description is usually simpler in textual
  - For structural information graphical is often better
- For simple languages
  - Form-based editing might also be an alternative

Xtext and GMF on the same instance model



Derived Graphical viewer support

- Xtext Generic Viewer component
  - Created by Xtext developers
  - Independent from the main Xtext development
  - Requies an extra language
    - to define uni-directional mapping
    - to define format
- See “A fresh look at graphical modeling” for details
  - http://www.slideshare.net/schwurbel/a-fresh-look-at-graphical-editing-10068461

Concrete Syntax Design

Conclusion

Concrete Syntax Design

- Multiple approaches
  - Textual and/or graphical syntaxes
  - Combinable
- Large amount of development work needed
  - Directly used by users
  - Usability issues
- Not everything is coded in an editor
  - Editor + corresponding views form the interface