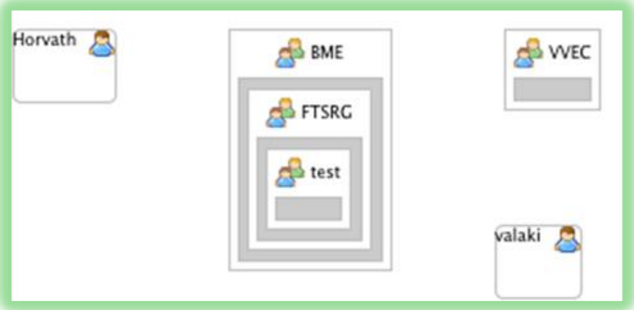
Concrete Syntax Design for Domain-specific Languages

Model Driven Software Development

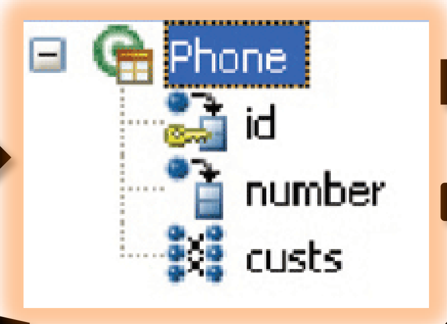
Lecture 5

Structure of DSMs

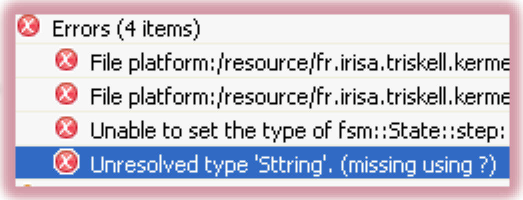
Graphical syntax



Abstract syntax



Well-formedness constraints

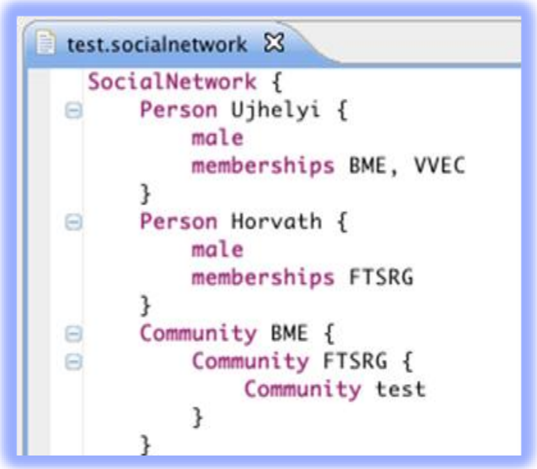


```
Errors (4 items)
  File platform:/resource/fr.irisa.triskell.kerme
  File platform:/resource/fr.irisa.triskell.kerme
  Unable to set the type of fsm::State::step:
  Unresolved type 'Sttring'. (missing using ?)
```

Behavioural semantics, simulation

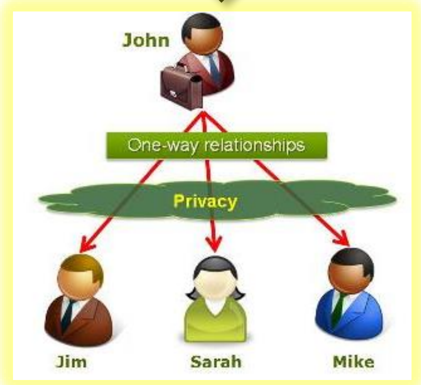
Mapping

Code generation

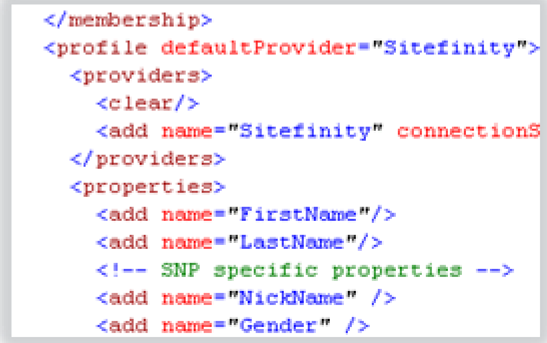


```
test.socialnetwork
SocialNetwork {
    Person Ujhelyi {
        male
        memberships BME, VVEC
    }
    Person Horvath {
        male
        memberships FTSRG
    }
    Community BME {
        Community FTSRG {
            Community test
        }
    }
```

Textual syntax



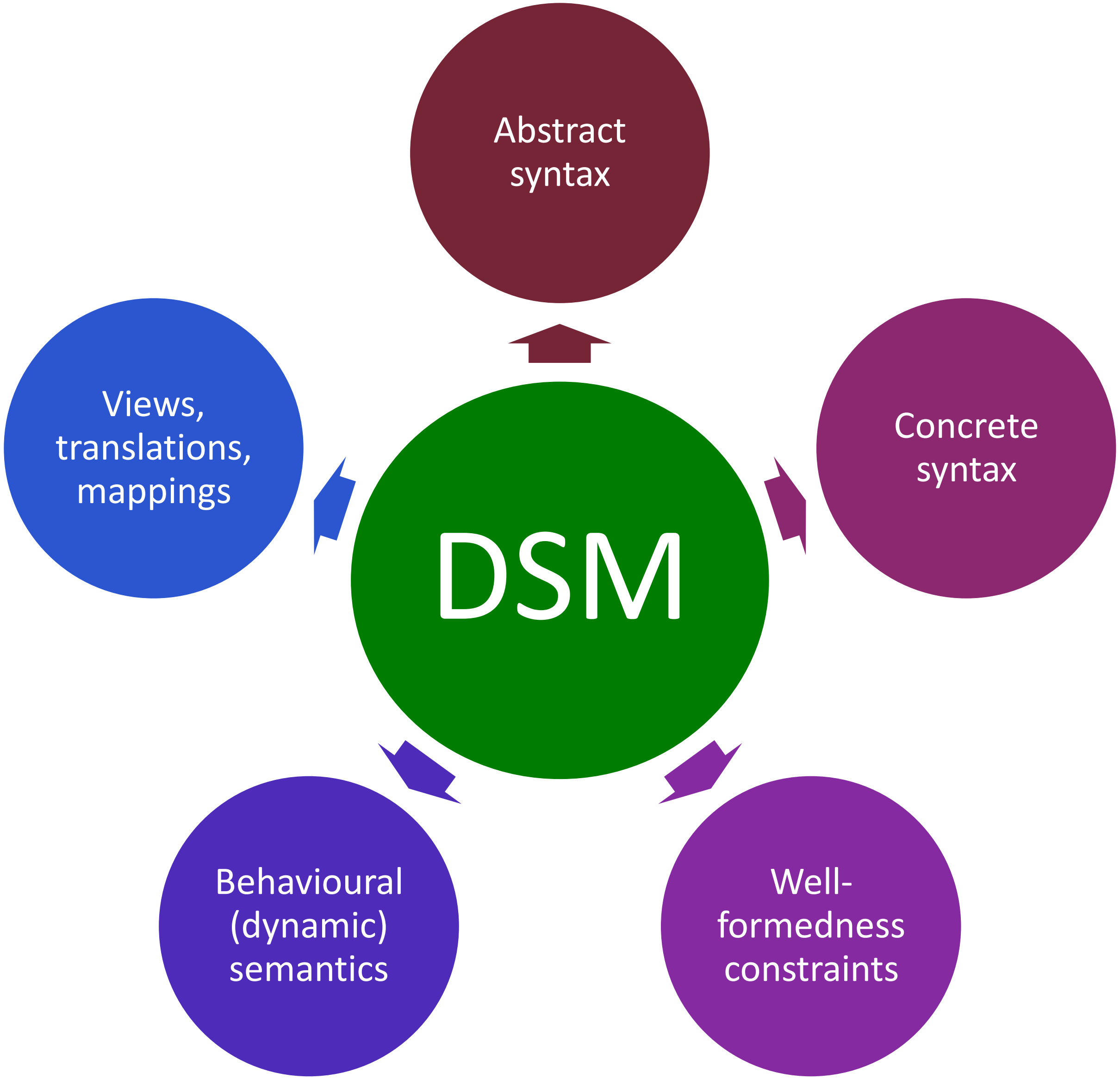
View



```
</membership>
<profile defaultProvider="Sitefinity">
  <providers>
    <clear/>
    <add name="Sitefinity" connectionS
  </providers>
  <properties>
    <add name="FirstName"/>
    <add name="LastName"/>
    <!-- SNP specific properties -->
    <add name="NickName" />
    <add name="Gender" />
```

Code (documentation, configuration)

DSM aspects



DSM

Abstract syntax

Concrete syntax

Well-formedness constraints

Behavioural (dynamic) semantics

Views, translations, mappings

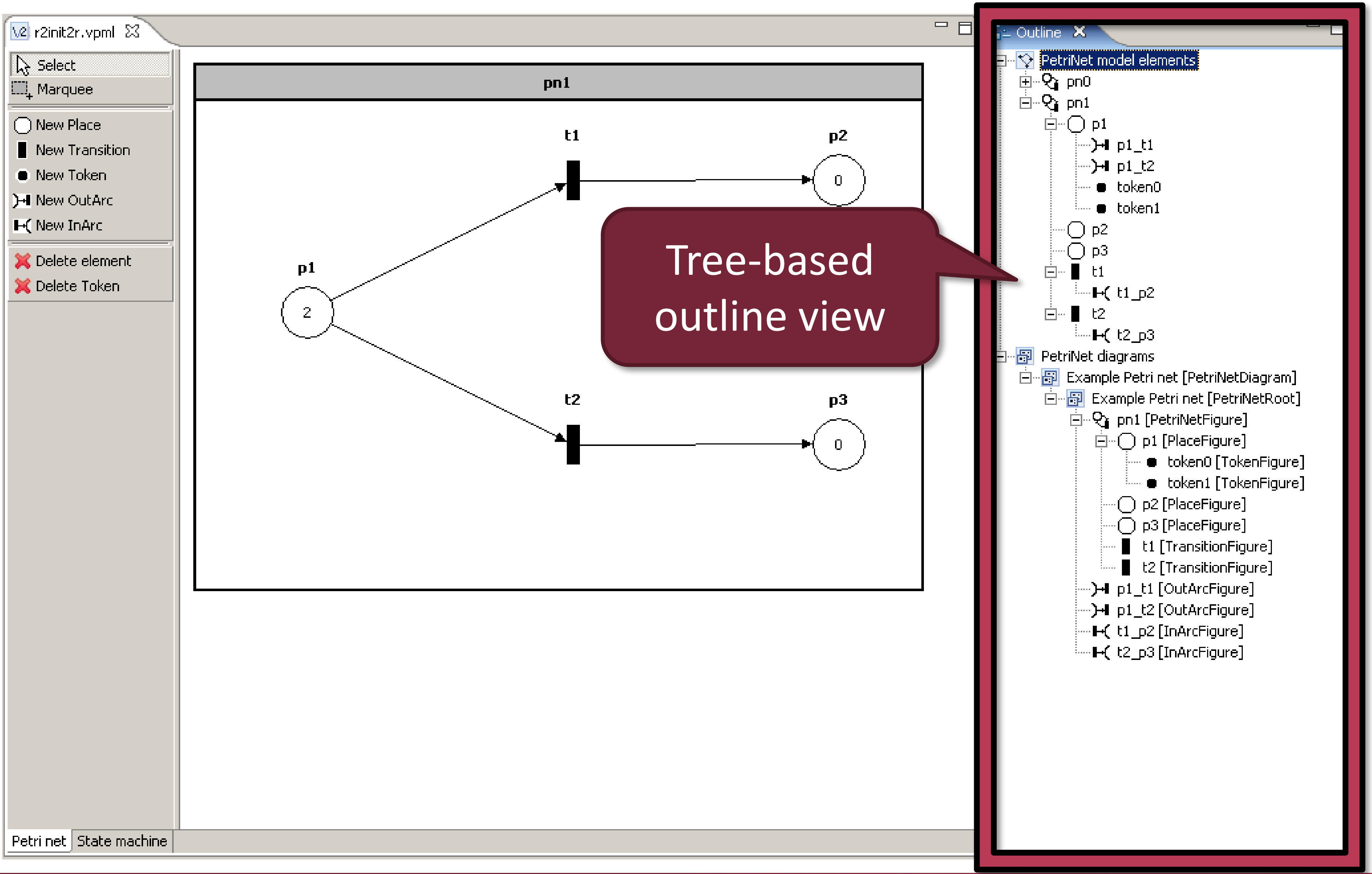
Concrete Syntax Design

- User-facing parts of a modeling language
  - Performance
  - Robustness
  - Usability issues
- Creating model editors
  - Similar problems as programming languages
  - IDE extensions needed
- Viewers are also important!
  - ~read-only editors

Concrete Syntax Approaches

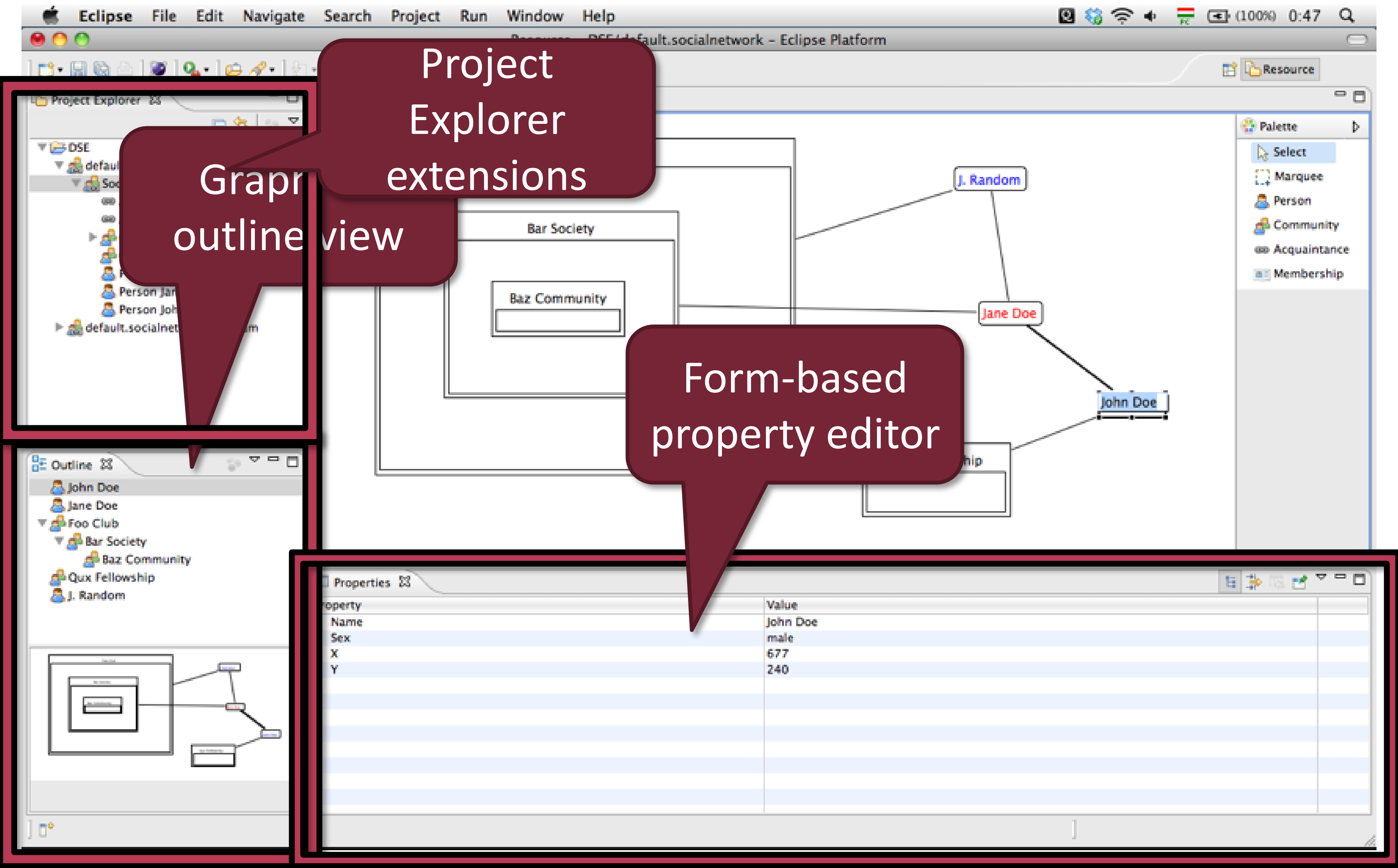
- Graphical
  - Focus of latter half of today’s lecture
  - Typically graph-based modeling (Edges, Nodes)
- Textual
  - More details to come in next lecture
- Form-based
  - Tree views
  - Property sheets, combo / radio /etc.
  - Table/matrix approaches

Example: Petri net editor



Tree-based outline view

Example: Social Network editor



Project Explorer extensions

Graph outline view

Form-based property editor

operty	Value
Name	John Doe
Sex	male
X	677
Y	240

Advanced features

Viewer features

- Outlining / folding / abstraction
- Details / documentation overlay (e.g. Javadoc, „code mining”)
- Validation / task / etc. overlay
- Search, navigability
- Auto layout/formatting/sorting

Editor features

- Templates/snippets/examples
- Guidance (content assist / snap)
- Composite operations/tools/refactorings
- Automatic fixes
- Undo&Redo, Transactionality

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
- JetBrains MPS

Human Aspects

Textual vs. Graphical

Visual Design

Layouting

Question: textual or graphical?

- No clear choice, just rules of thumb

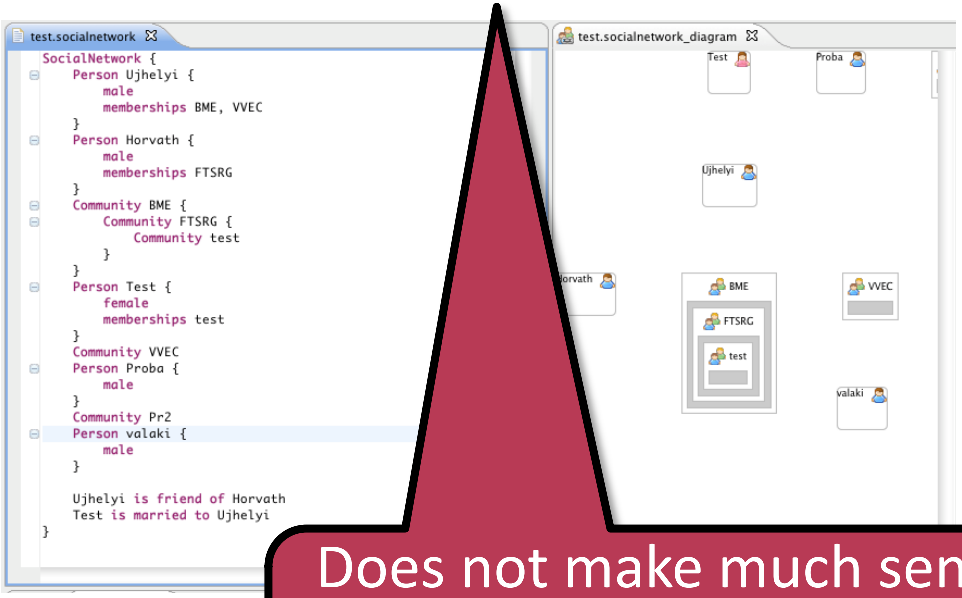
Textual Languages (*raw editing*)	Graphical Languages
Quick and simple editing	More cumbersome editing
References as *string identifiers*	References displayed visually
Inconsistent during editing	Always syntactically correct
Trivial diff&patch, copy&paste, search&replace	Editing services require tool development effort
Typically better for behavior	Typically better for structure

  - Simple languages: consider form-based as well
    - Like graphical, but cross-references poorly supported

- …why not both?

Textual + Graphical

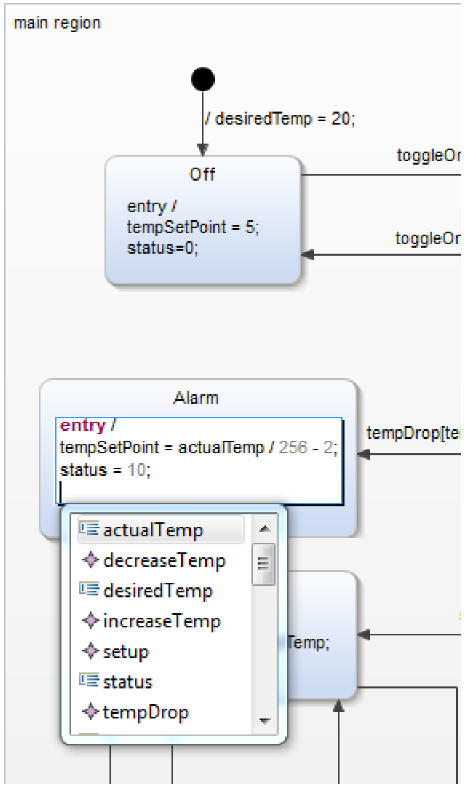
- Same model, two syntaxes
  - Text editor + graphical view ☺
    - Xtext Generic Viewer
  - Textual + graphical editors
    - Xtext + GMF side-by-side
- Different aspects of model
  - Diagram with text fields
    - Embedded Xtext support



test.socialnetwork

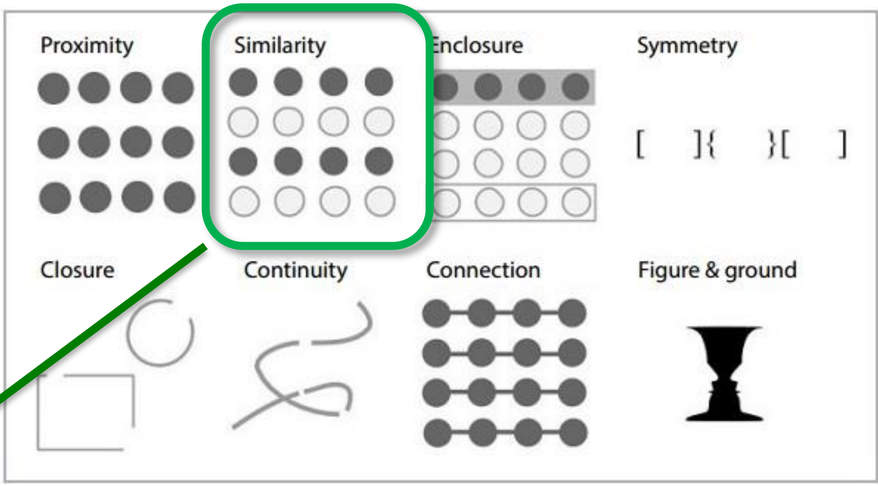
```
SocialNetwork {
    Person Ujhelyi {
        male
        memberships BME, VVEC
    }
    Person Horvath {
        male
        memberships FTSRG
    }
    Community BME {
        Community FTSRG {
            Community test
        }
    }
    Person Test {
        female
        memberships test
    }
    Community VVEC
    Person Proba {
        male
    }
    Community Pr2
    Person valaki {
        male
    }

    Ujhelyi is friend of Horvath
    Test is married to Ujhelyi
}
```

Does not make much sense, don't do this in the homework!

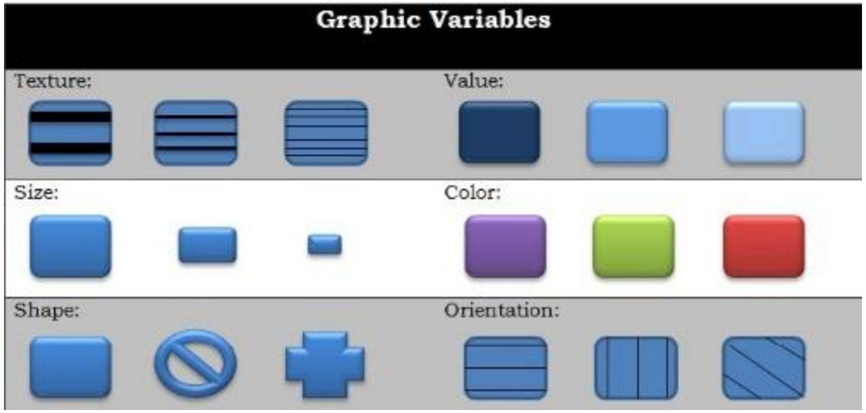


Visual Design 101

- What belongs together? „Gestalt principles of grouping”
    - E.g. which label belongs to which node?



- What is similar? „Bertin’s visual variables”
    - Size, shape
    - Color hue, value, intensity
    - Line style / orientation / texture

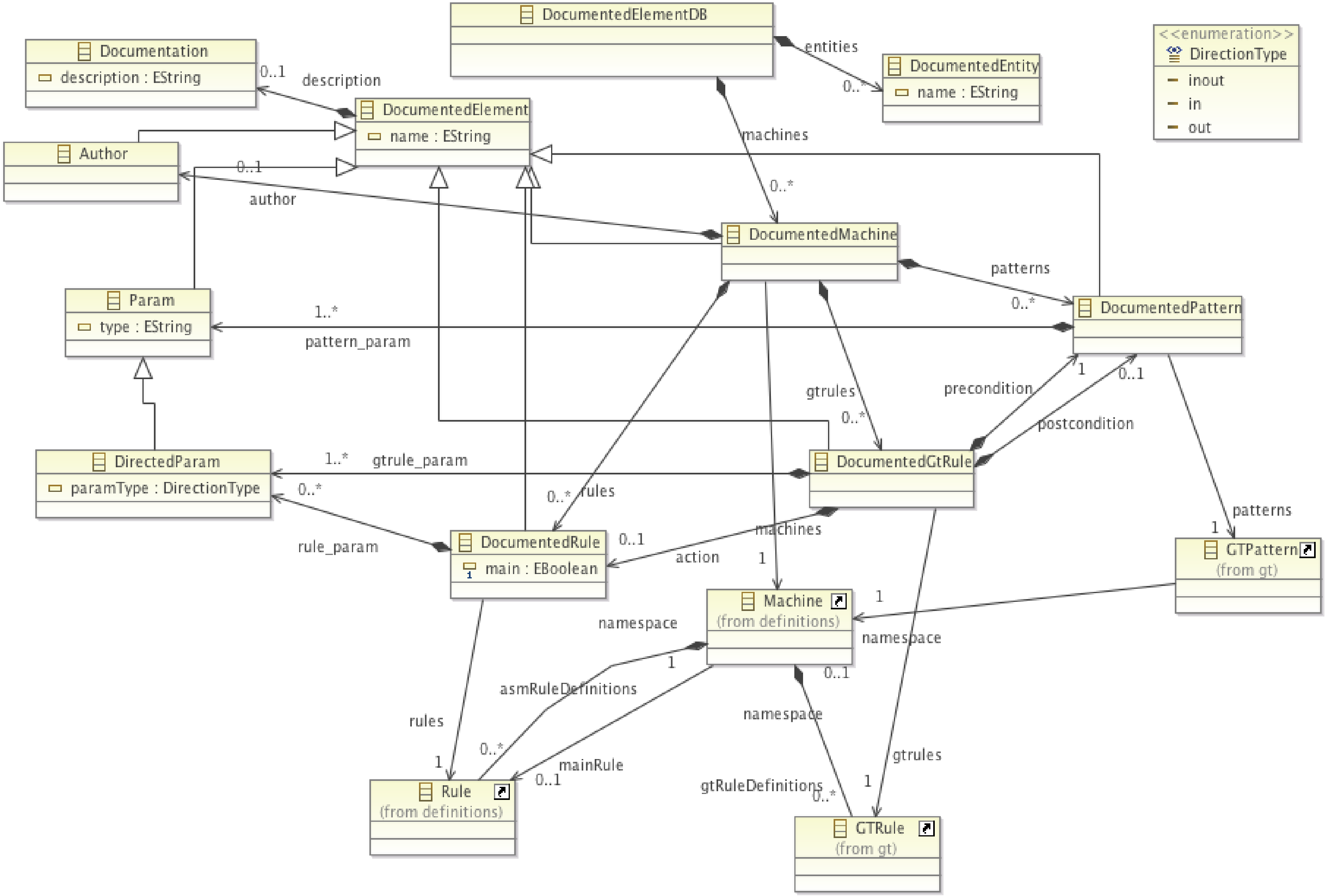


Sources: http://wiki.gis.com/wiki/index.php/Visual_variable
https://www.fusioncharts.com/blog/how-to-use-the-gestalt-principles-for-visual-storytelling-podv/

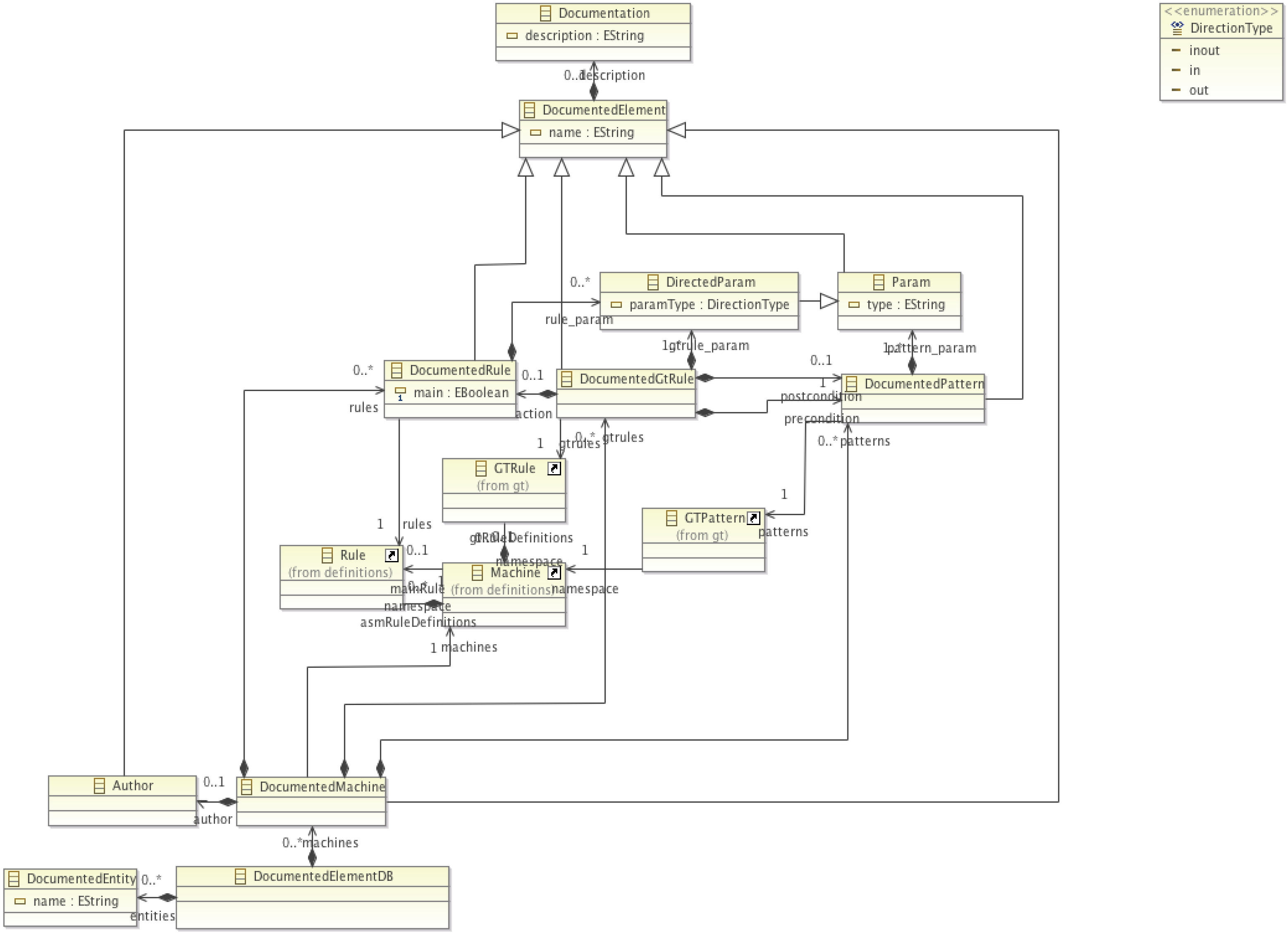
Scaling issues

- Cumbersome editing
  - E.g., automatically reorganize diagram when inserting 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

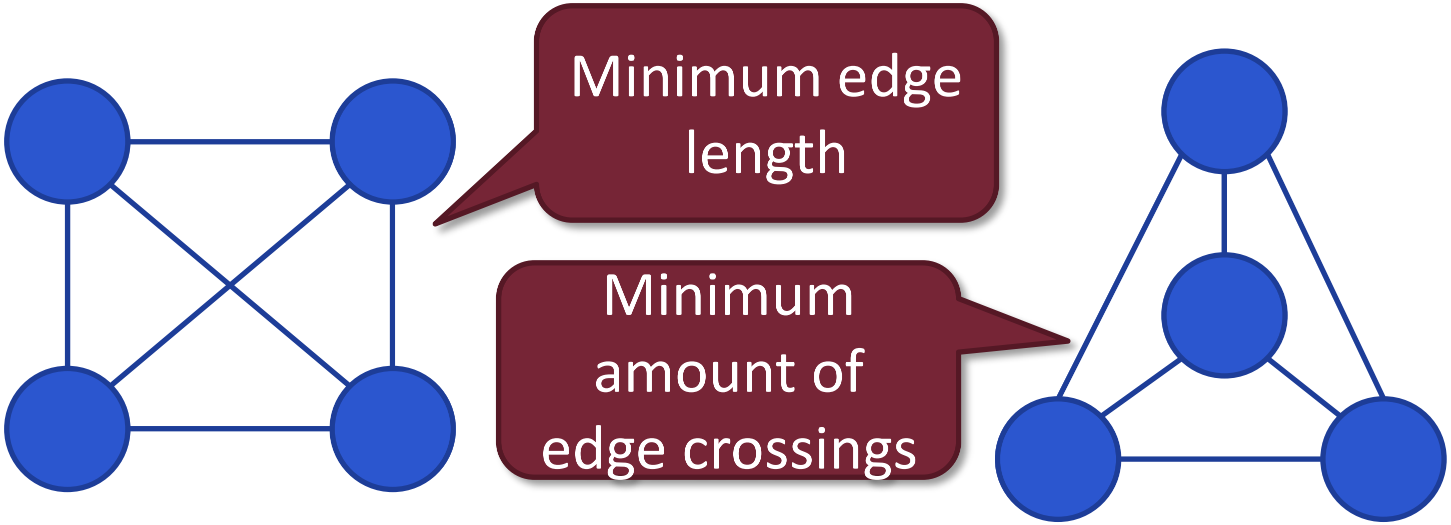


Example: Layouting



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

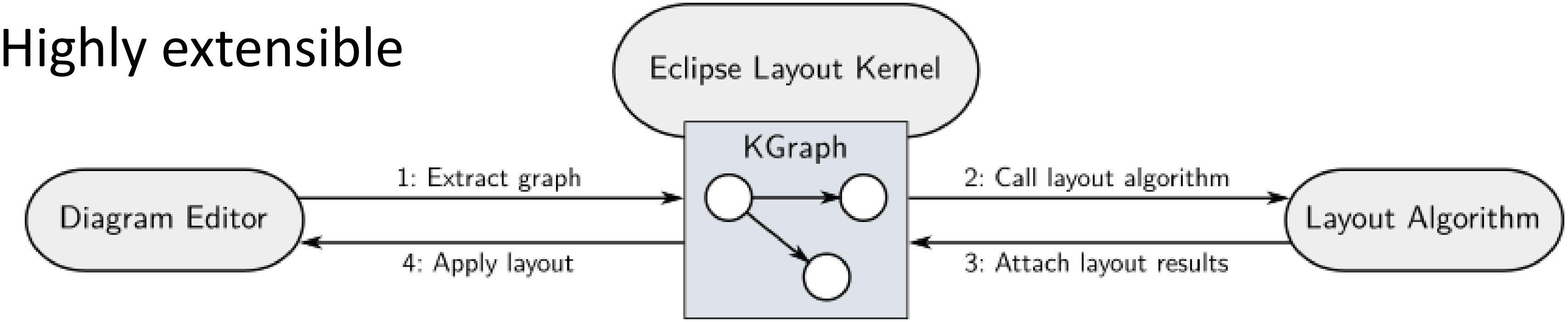


Minimum edge length

Minimum amount of edge crossings

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



Editor Engineering

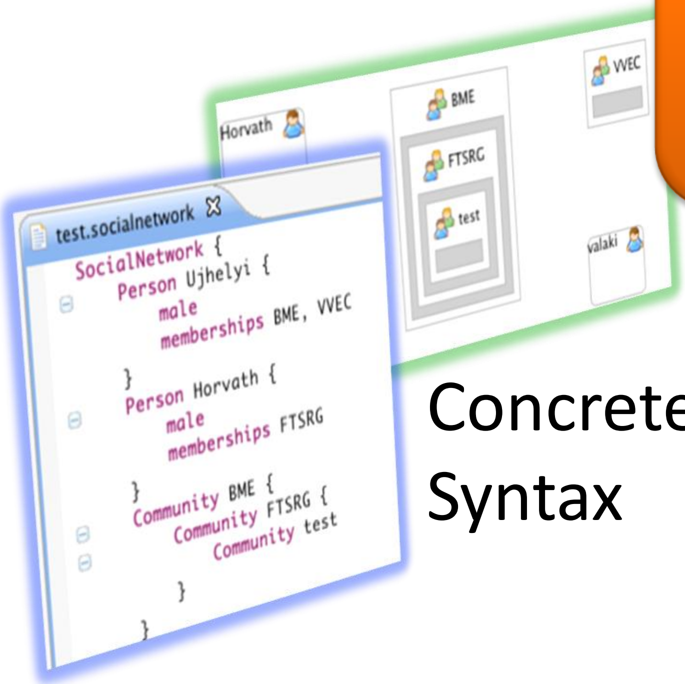
Editing Workflows
Transactionality
Notation Models

Projectional vs Raw

- Workflow 1: **projectional editing**
  - AKA syntax-driven editing, structural editing

Edit abstract representation

- Insert model element
- Remove model element
- Insert reference
- Remove reference
- Modify attribute

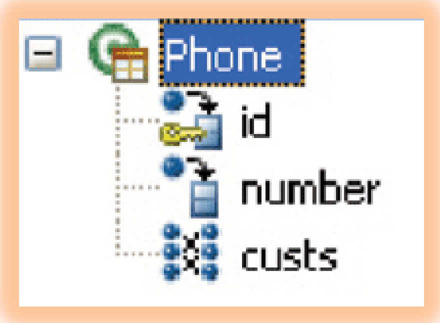


```
test.socialnetwork
SocialNetwork {
  Person Ujhelyi {
    male
    memberships BME, VVEC
  }
  Person Horvath {
    male
    memberships FTSRG
  }
  Community BME {
    Community FTSRG {
      Community test
    }
  }
}
```

Concrete Syntax

2

1



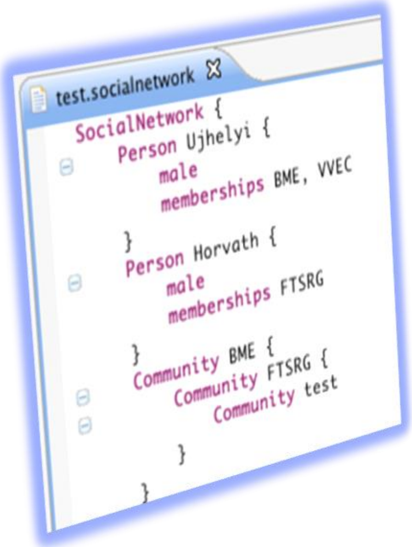
Model

Derive / project concrete representation

- Pretty print (textual)
- Visualize / layout (graphical)

Projectional vs Raw

- Workflow 2: **raw editing** (w. textual syntax)
  - AKA source editing



test.socialnetwork

```
SocialNetwork {
    Person Ujhelyi {
        male
        memberships BME, VVEC
    }
    Person Horvath {
        male
        memberships FTSRG
    }
    Community BME {
        Community FTSRG {
            Community test
        }
    }
}
```

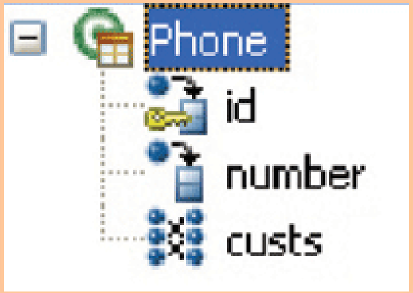
Concrete Syntax

Edit concrete representation

- Insert character(s)
- Delete character(s)
- Replace character(s)

Derive abstract representation

- Parse textual format

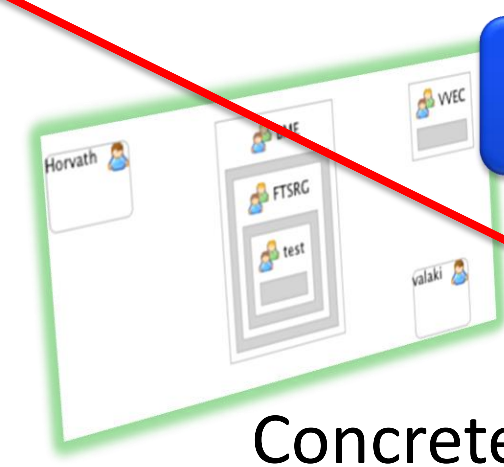


Model

Projectional vs Raw

- Workflow 2: **raw editing** (w. graphical syntax)

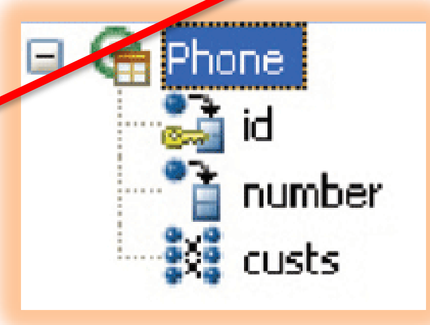
Highly impractical



Edit concrete representation

- Paint diagram (e.g. PNG)

1 → 2



Concrete Syntax

Model

Derive abstract representation

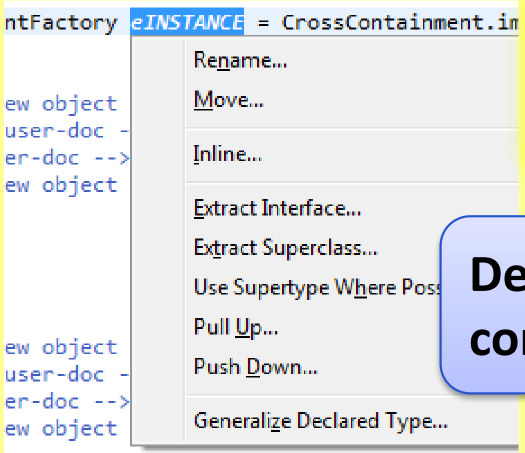
- Image processing with convolutional neural nets
- OCR, etc.

Projectional vs Raw

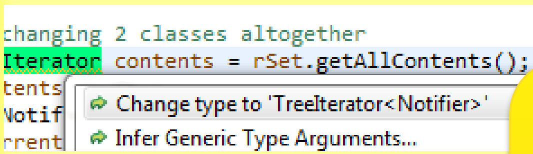
- „Feature matrix” + examples

	Graphical syntax	Textual syntax
Raw editing	✗	Typical Xtext
Projectional editing	Typical Sirius	Rare MPS

Mixed workflow



```
ntFactory eINSTANCE = CrossContainment.in

ew object
user-doc -
er-doc -->
ew object

ew object
user-doc -
er-doc -->
ew object
```



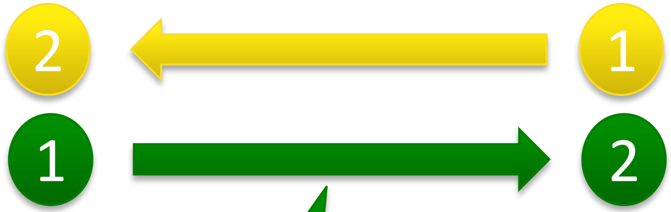
```
changing 2 classes altogether
Iterator contents = rSet.getAllContents();
tents
Notif
rrent
```

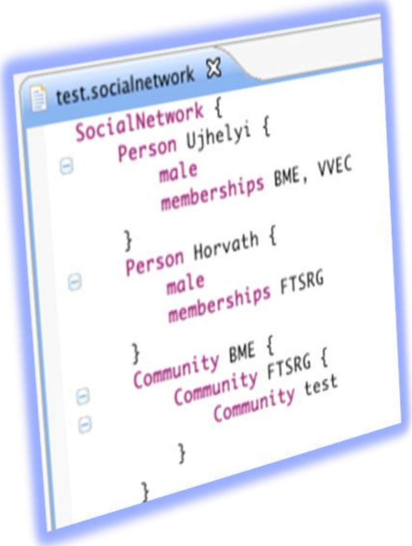
Change type to 'TreeIterator<Notifier>'
Infer Generic Type Arguments...

Complex manipulation of abstract representation

- Quick fix
- Refactor
- M2M

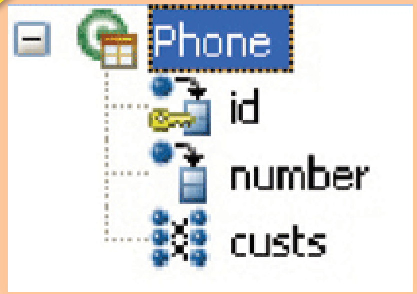
Derive / project concrete representation





```
test.socialnetwork
SocialNetwork {
  Person Ujhelyi {
    male
    memberships BME, VVEC
  }
  Person Horvath {
    male
    memberships FTSRG
  }
  Community BME {
    Community FTSRG {
      Community test
    }
  }
}
```

Concrete Syntax

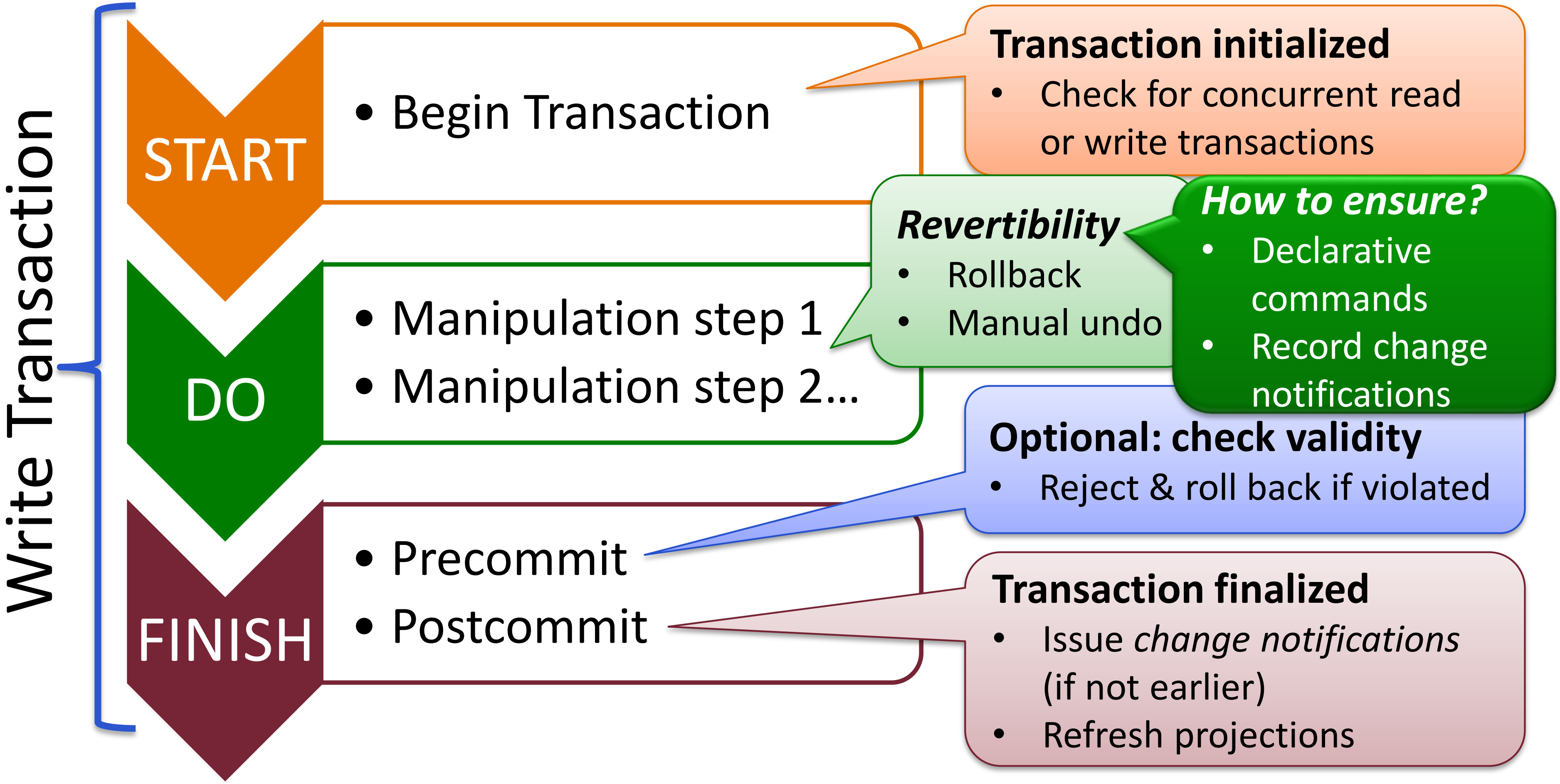


Model

Normal raw editing workflow

Transactions in projectional editing

- Complex manipulation sequence as single action
  - „Extract subprocess”, „Drag&drop attribute” etc.



Write Transaction

START

- Begin Transaction

Transaction initialized

- Check for concurrent read or write transactions

DO

- Manipulation step 1
- Manipulation step 2…

Revertibility

- Rollback
- Manual undo

How to ensure?

- Declarative commands
- Record change notifications

FINISH

- Precommit
- Postcommit

Optional: check validity

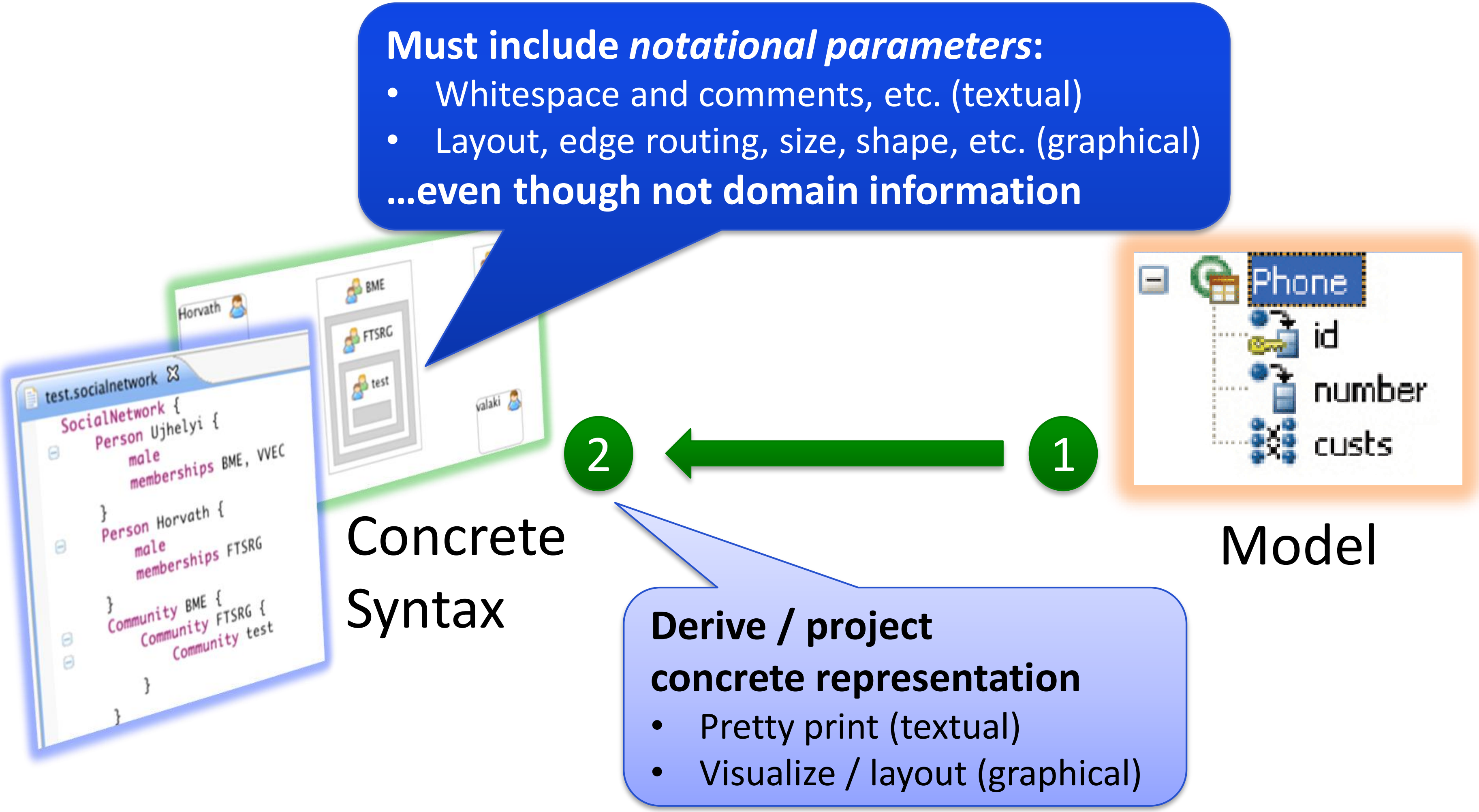
- Reject & roll back if violated

Transaction finalized

- Issue *change notifications* (if not earlier)
- Refresh projections

Superfluous notational parameters

- Workflow 1: **projectional editing**



Must include *notational parameters*:

- Whitespace and comments, etc. (textual)
- Layout, edge routing, size, shape, etc. (graphical)

…even though not domain information

test.socialnetwork

```
SocialNetwork {
    Person Ujhelyi {
        male
        memberships BME, VVEC

    }
    Person Horvath {
        male
        memberships FTSRG

    }
    Community BME {
        Community FTSRG {
            Community test

        }
    }
```

Derive / project concrete representation

- Pretty print (textual)
- Visualize / layout (graphical)

Deriving notational parameters

- Notational parameters can be…
  - …”baked into” projection code
    - e.g. all lines are black, all fonts are 10pt (graphical)
    - e.g. apply this code formatting template (textual)
  - …derived from domain information
    - e.g. shape determined by type, color by visibility

Problem 1:
Editable parameters cannot be a function of the domain model, must be stored

Problem 2:
Providing sane values is difficult for some parameters e.g. position in diagram

  - **…stored in the model**

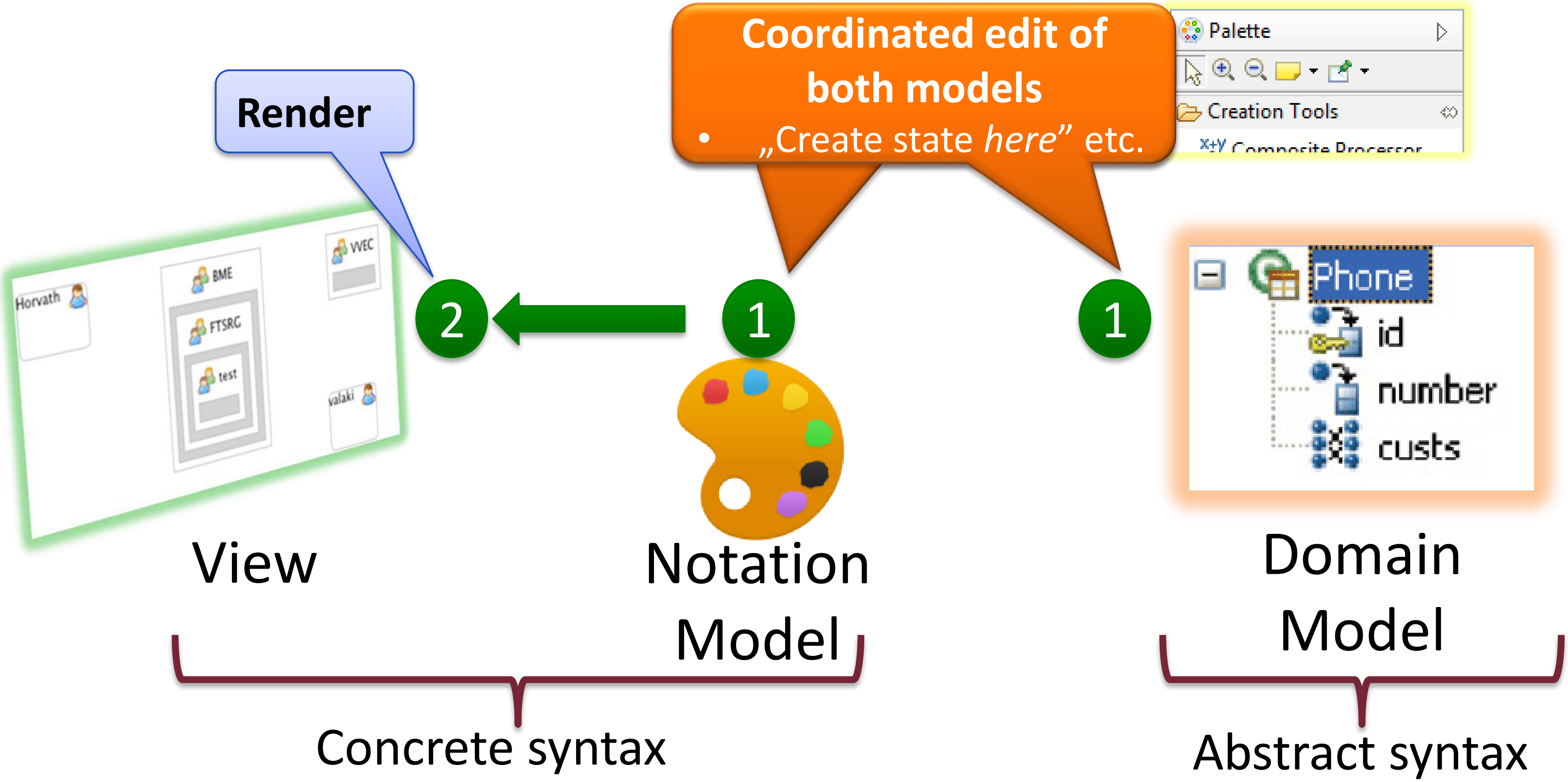
Notation/view models

M.Fowler's „Presentation Model" architectural pattern

- Decompose model:
  - Domain / Semantic model (abstract syntax)
  - **Notation model** (view model): presentation state
    - may be editable by user
    - but still needs derivable defaults → see layouting
- Generic implementation in GMF and Graphiti
  - Based on EMF, in fact
- Often stored in external files
  - Separation of concerns
  - E.g. code generator not interested in view information

Editing workflow with notation models

- Workflow 1: **projectional editing**
  - Scenario A: co-modifying domain&notation models



Coordinated edit of both models

- „Create state *here*” etc.

Render

View

Notation Model

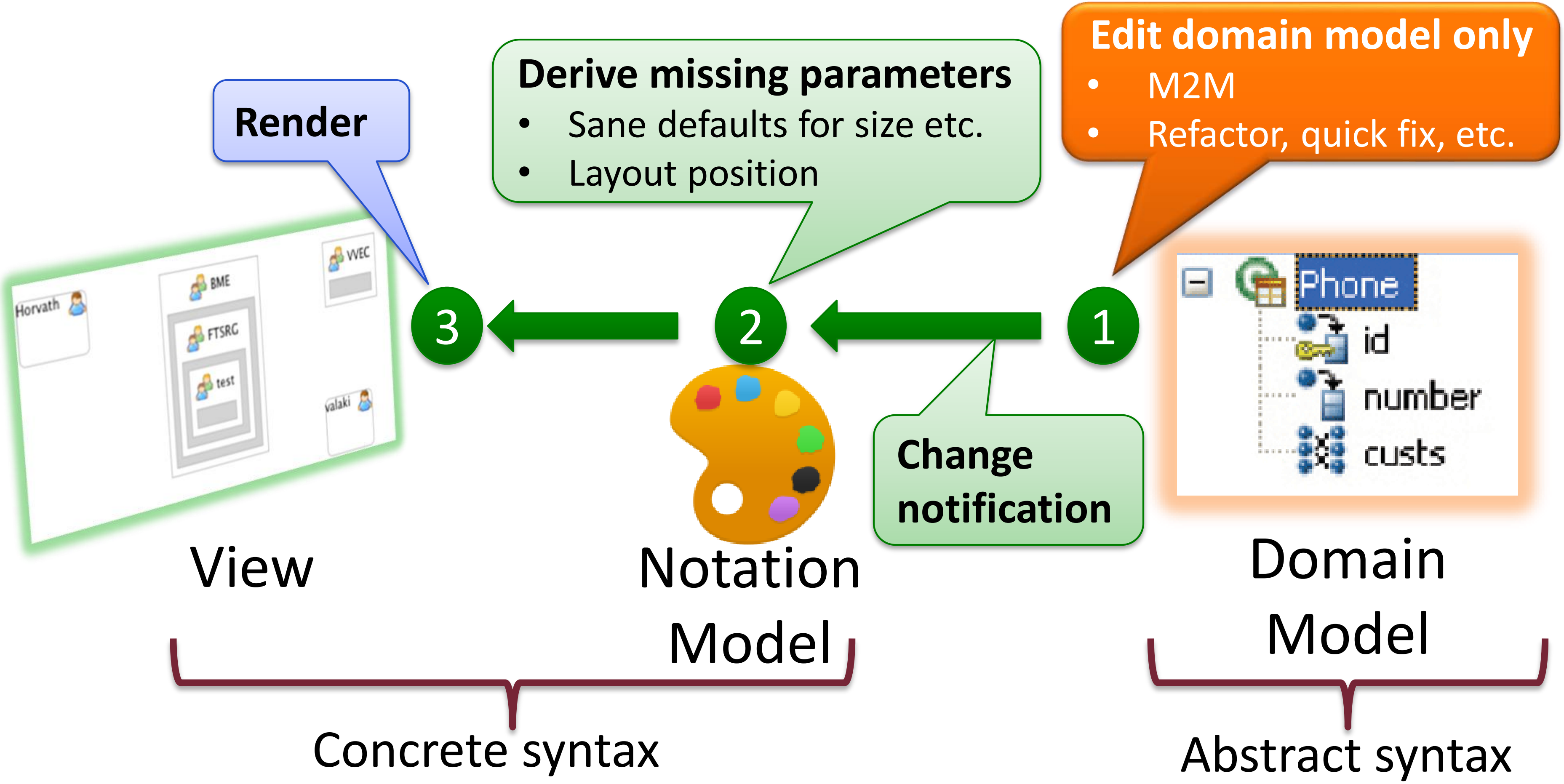
Domain Model

Concrete syntax

Abstract syntax

Editing workflow with notation models

- Workflow 1: **projectional editing**
  - Scenario B: modifying domain model only



Edit domain model only
- M2M
- Refactor, quick fix, etc.

Derive missing parameters
- Sane defaults for size etc.
- Layout position

Render

Change notification

View

Notation Model

Domain Model

Concrete syntax

Abstract syntax

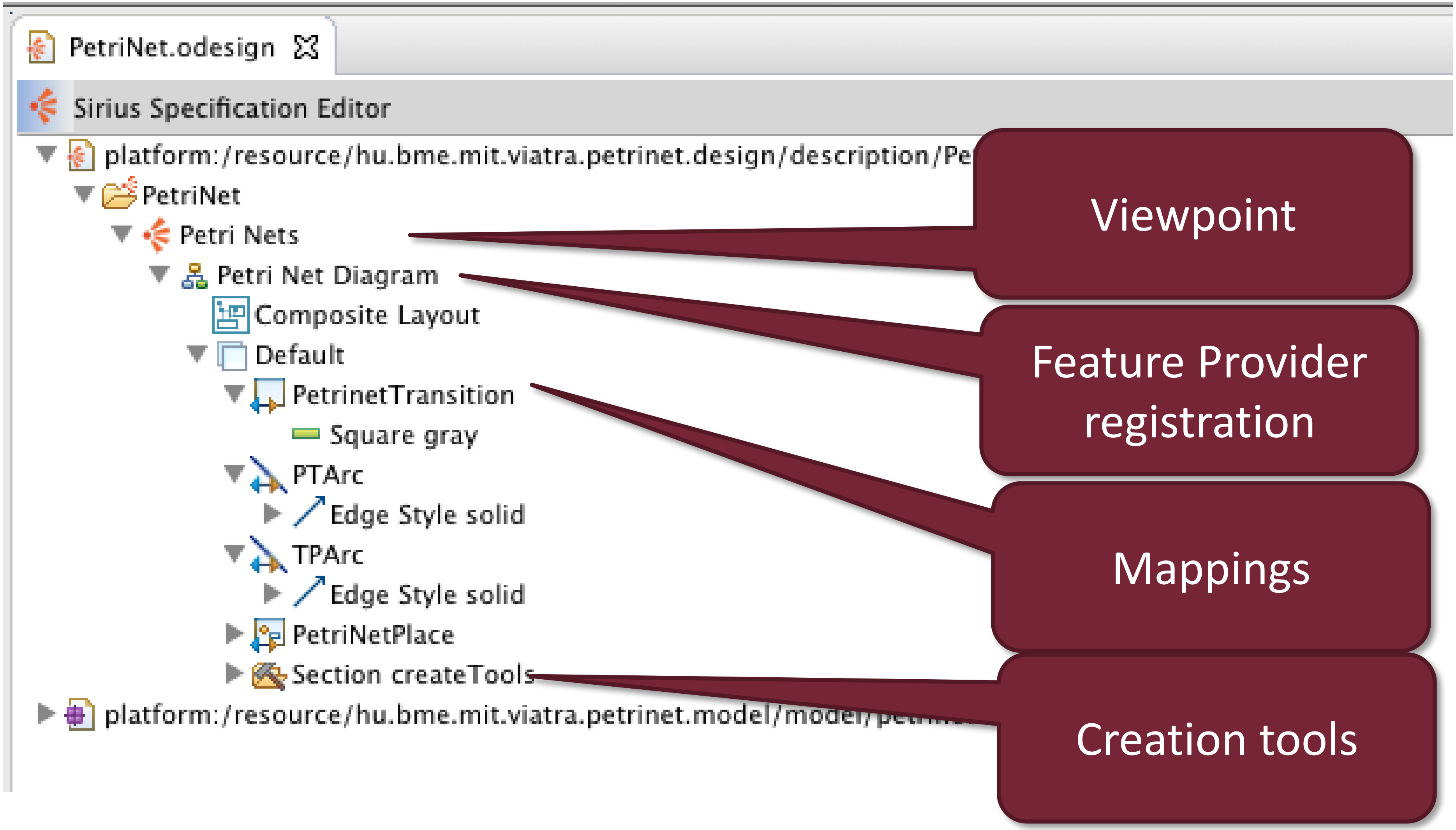
Eclipse Sirius

Sirius

Sirius Viewpoints

- Base concept:
  - Viewpoints for different roles
  - Every editor/viewer 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

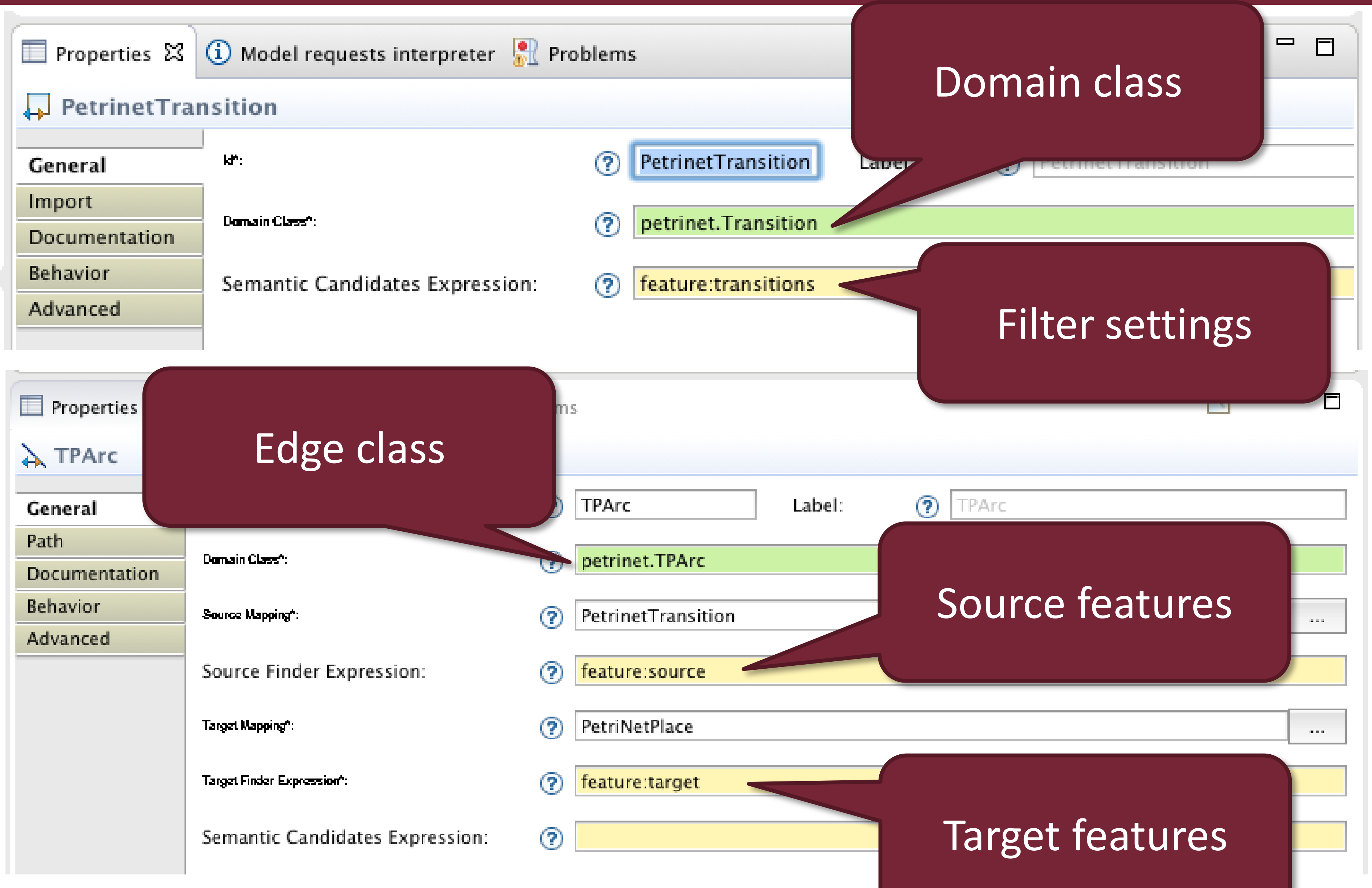


PetriNet.odesign

Sirius Specification Editor

```
▼ platform:/resource/hu.bme.mit.viatra.petrinet.design/description/Pe
  ▼ PetriNet
    ▼ Petri Nets
      ▼ Petri Net Diagram
          Composite Layout
        ▼ Default
          ▼ PetrinetTransition
              Square gray
          ▼ PTArc
            ▶ Edge Style solid
          ▼ TPArc
            ▶ Edge Style solid
          ▶ PetriNetPlace
          ▶ Section createTools
▶ platform:/resource/hu.bme.mit.viatra.petrinet.model/model/petrin
```

Node & Edge Mapping



PetrinetTransition

Id*: PetrinetTransition

Label: PetrinetTransition

Domain Class*: petrinet.Transition

Semantic Candidates Expression: feature:transitions

TPArc

TPArc

Label: TPArc

Domain Class*: petrinet.TPArc

Source Mapping*: PetrinetTransition

Source Finder Expression: feature:source

Target Mapping*: PetriNetPlace

Target Finder Expression*: feature:target

Semantic Candidates Expression:

Domain class

Filter settings

Edge class

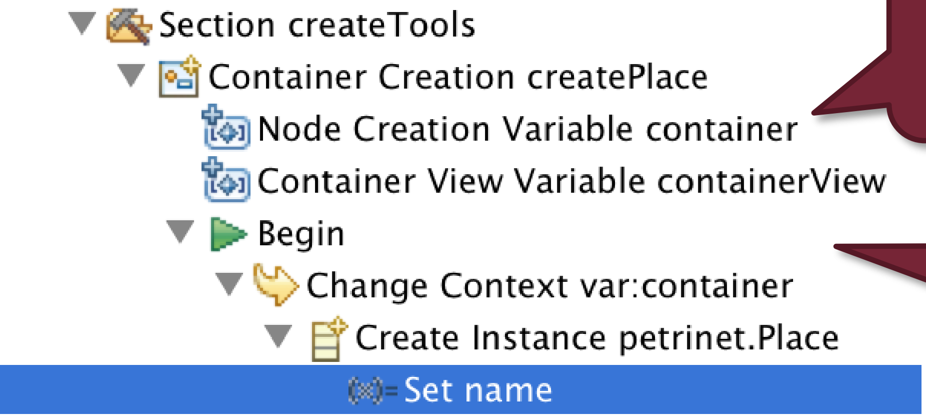
Source features

Target features

Feature Selection

- Interpreted **model query** 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

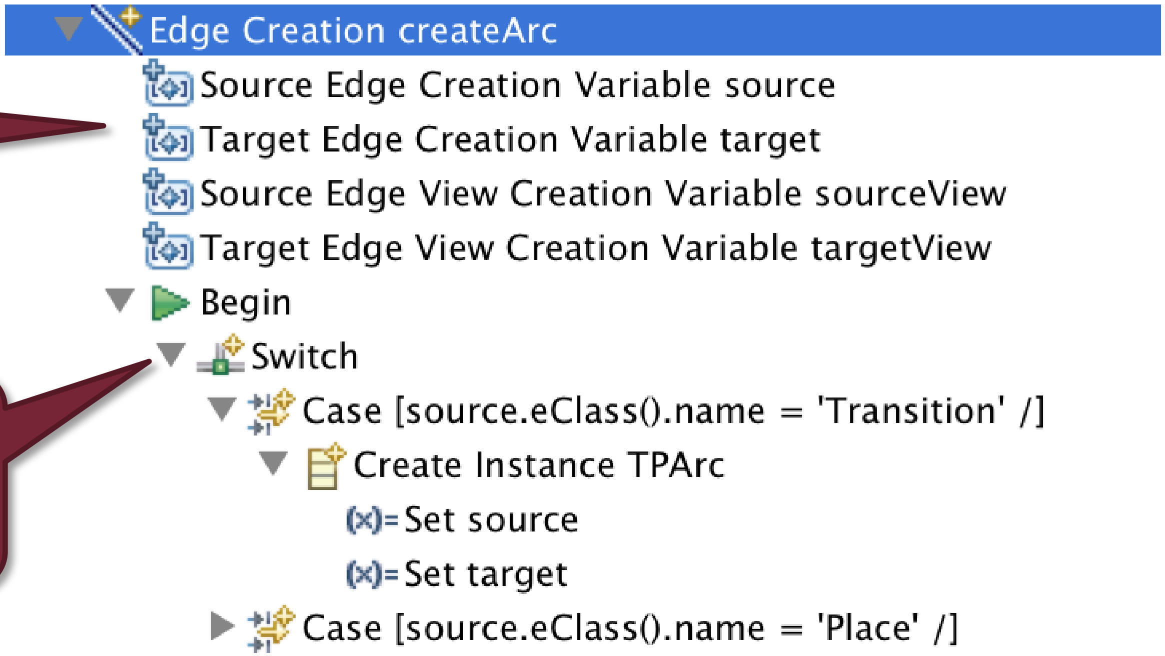
Node & Edge Tool



- Section createTools
  - Container Creation createPlace
    - Node Creation Variable container
    - Container View Variable containerView
    - Begin
      - Change Context var:container
        - Create Instance petrinet.Place
          - Set name

Tool parameter variables

Model creation sequence



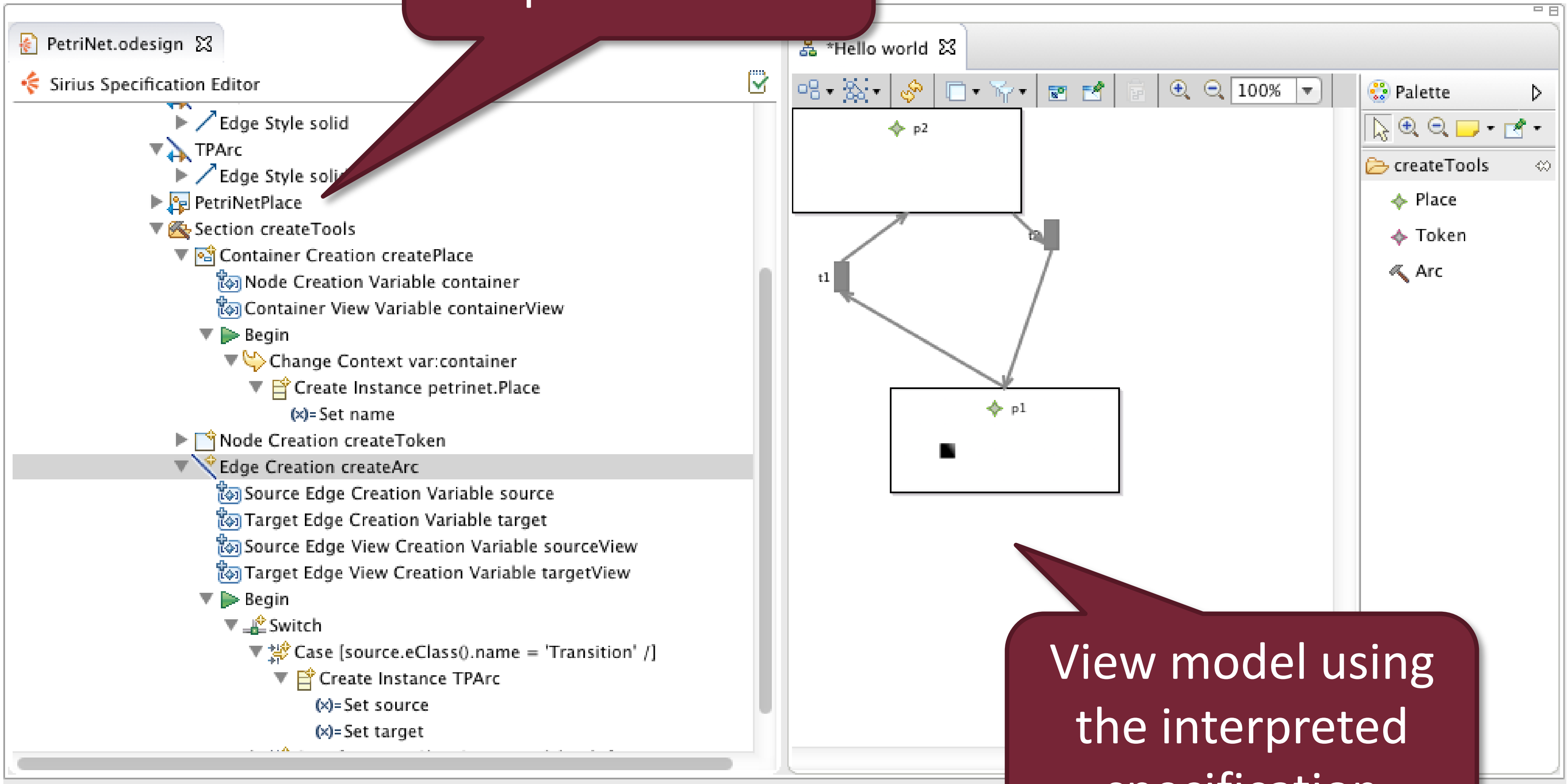
- Edge Creation createArc
  - Source Edge Creation Variable source
  - Target Edge Creation Variable target
  - Source Edge View Creation Variable sourceView
  - Target Edge View Creation Variable targetView
  - Begin
    - Switch
      - Case [source.eClass().name = 'Transition' /]
        - Create Instance TPArc
          - Set source
          - Set target
      - Case [source.eClass().name = 'Place' /]

Different variables

More complex creation steps

Interpreted Modeler Development

Viewpoint specification



PetriNet.odesign

Sirius Specification Editor

- Edge Style solid
- TPArc
  - Edge Style solid
- PetriNetPlace
- Section createTools
  - Container Creation createPlace
    - Node Creation Variable container
    - Container View Variable containerView
    - Begin
      - Change Context var:container
        - Create Instance petrinet.Place
          - Set name
  - Node Creation createToken
  - Edge Creation createArc
    - Source Edge Creation Variable source
    - Target Edge Creation Variable target
    - Source Edge View Creation Variable sourceView
    - Target Edge View Creation Variable targetView
    - Begin
      - Switch
        - Case [source.eClass().name = 'Transition' /]
          - Create Instance TPArc
            - Set source
            - Set target

View model using the interpreted specification

„Hot topic”: Language Servers

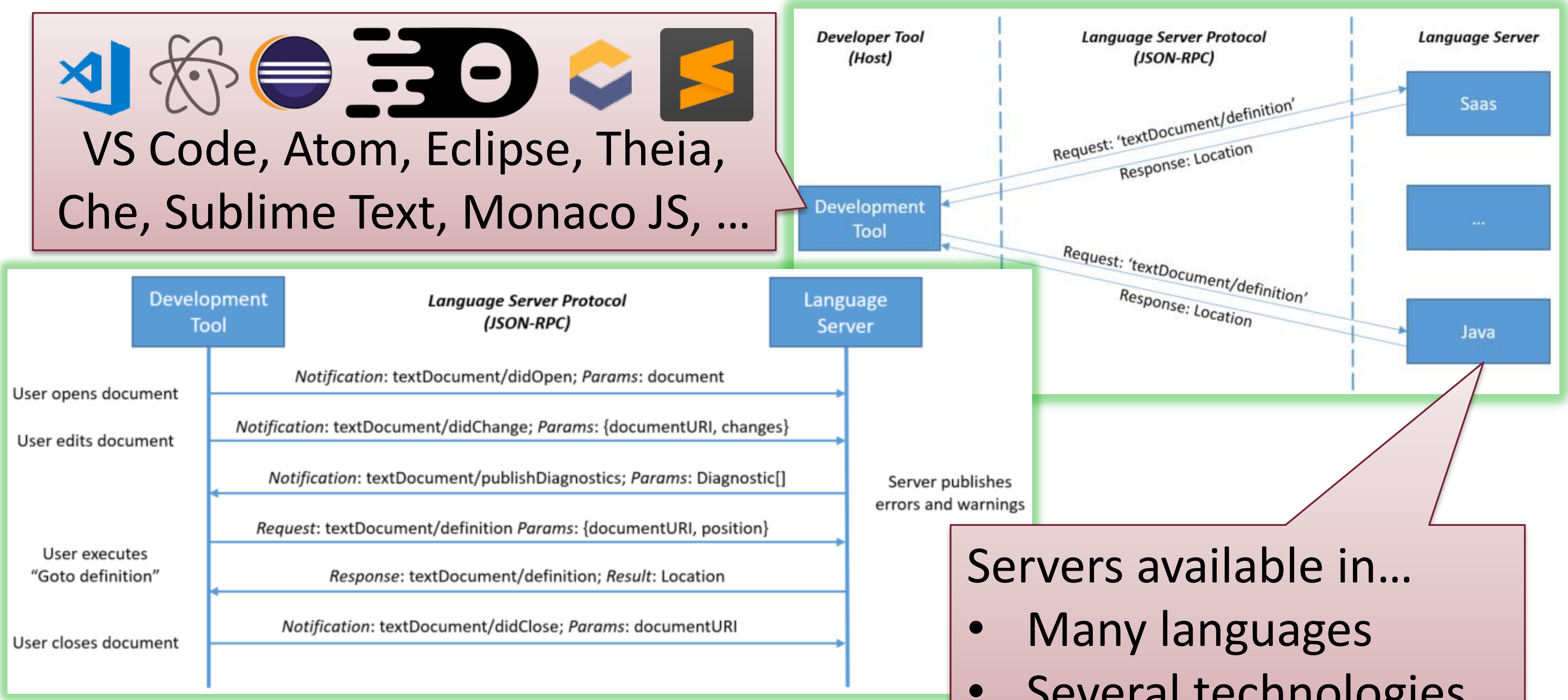
Language Server Protocol (LSP)

Graphical Language Server Protocol (GLSP)

Language Server Protocol (LSP)

- Delegate some editor services to language server
  - Protocol originally by Microsoft, for VS Code
    - (standardized since 2016)

VS Code, Atom, Eclipse, Theia, Che, Sublime Text, Monaco JS, …


Developer Tool (Host)
Language Server Protocol (JSON-RPC)
Language Server
Saas
…
Java
Development Tool
Request: 'textDocument/definition'
Response: Location
Request: 'textDocument/definition'
Response: Location
Development Tool
Language Server Protocol (JSON-RPC)
Language Server
User opens document
Notification: textDocument/didOpen; Params: document
User edits document
Notification: textDocument/didChange; Params: {documentURI, changes}
Notification: textDocument/publishDiagnostics; Params: Diagnostic[]
Server publishes errors and warnings
User executes "Goto definition"
Request: textDocument/definition Params: {documentURI, position}
Response: textDocument/definition; Result: Location
User closes document
Notification: textDocument/didClose; Params: documentURI

Servers available in…
- Many languages
- Several technologies

What is delegated?

- Language services for textual languages
  - Semantic services: on language server only
    - Information overlay: hover, diagnostics…
    - Navigation: jump to, find, …
    - Editing: completion, refactor, …
  - Syntactic services: mixed
    - On language server: outline, folding
    - In IDE: syntax highlight **NOT** delegated

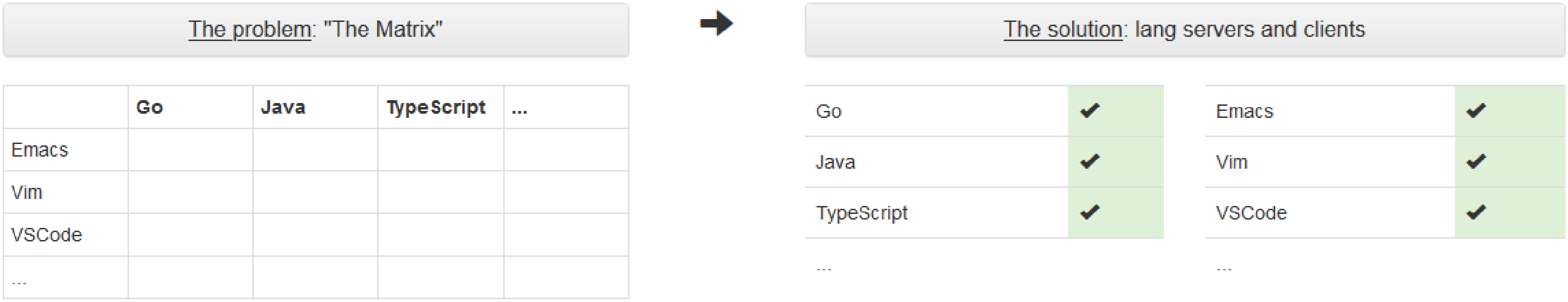
Still limited, but rapidly evolving

Still needs a language-specific IDE plugin!

- (GLSP for graphical languages)
    - Language-specific editor plugin still required!

Why is it delegated?

- Why is this better than just extensible IDEs?
  - To quote LSP docs:



The problem: "The Matrix"

	Go	Java	TypeScript	...
Emacs				
Vim				
VSCode				
...				

➜

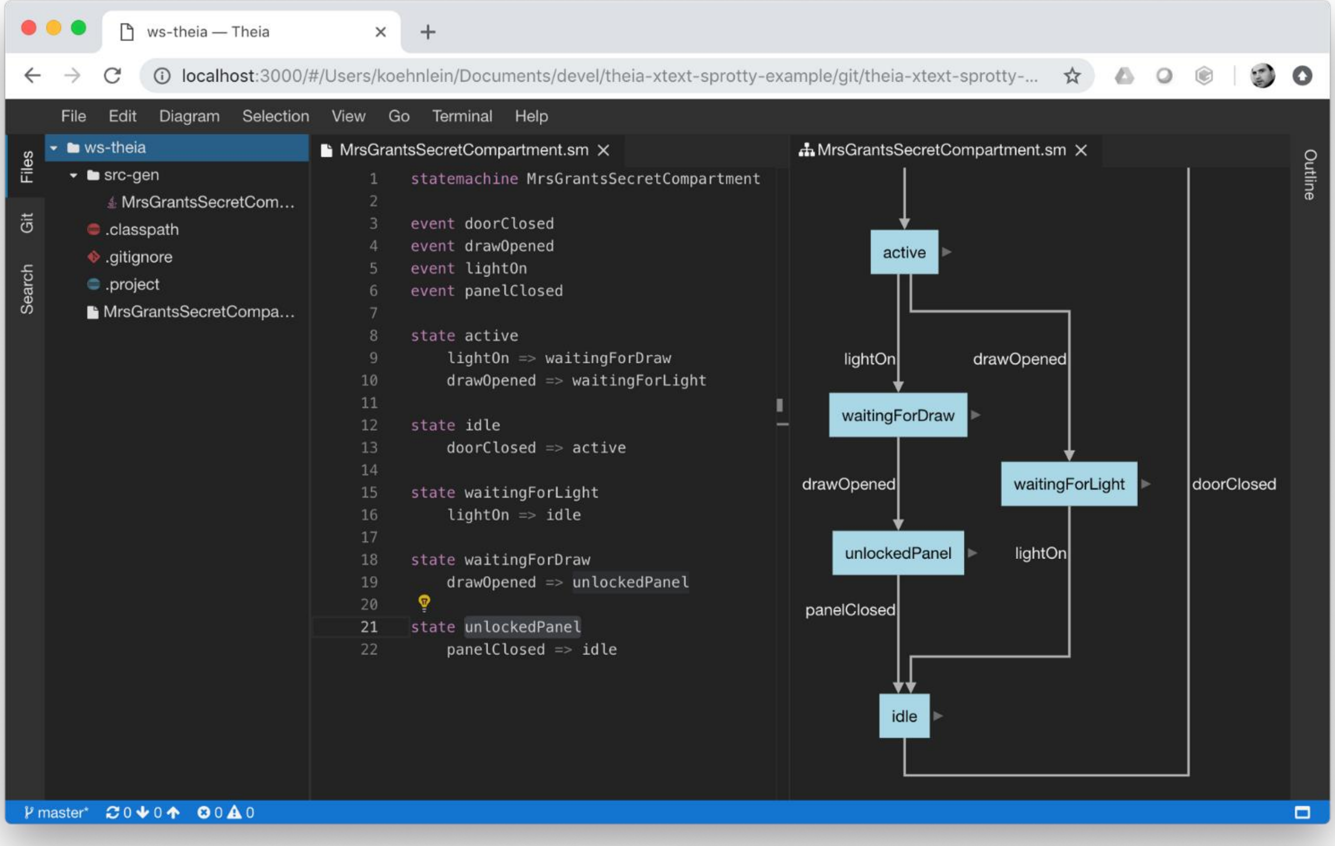
The solution: lang servers and clients

Go	✓
Java	✓
TypeScript	✓
...	

Emacs	✓
Vim	✓
VSCode	✓
...	

  - So to reuse *language-specific semantic services* as well
- Also: RPC → cross-platform integration
- Also: Cloud IDEs (see Eclipse Theia, Che, gitpod.io)
  - Easier provisioning per developer seat
  - “Thin clients”, resource-intensive services on the cloud

LSP (+ GLSP*) demo in Eclipse Theia



```
statemachine MrsGrantsSecretCompartment

event doorClosed
event drawOpened
event lightOn
event panelClosed

state active
    lightOn => waitingForDraw
    drawOpened => waitingForLight

state idle
    doorClosed => active

state waitingForLight
    lightOn => idle

state waitingForDraw
    drawOpened => unlockedPanel

state unlockedPanel
    panelClosed => idle
```

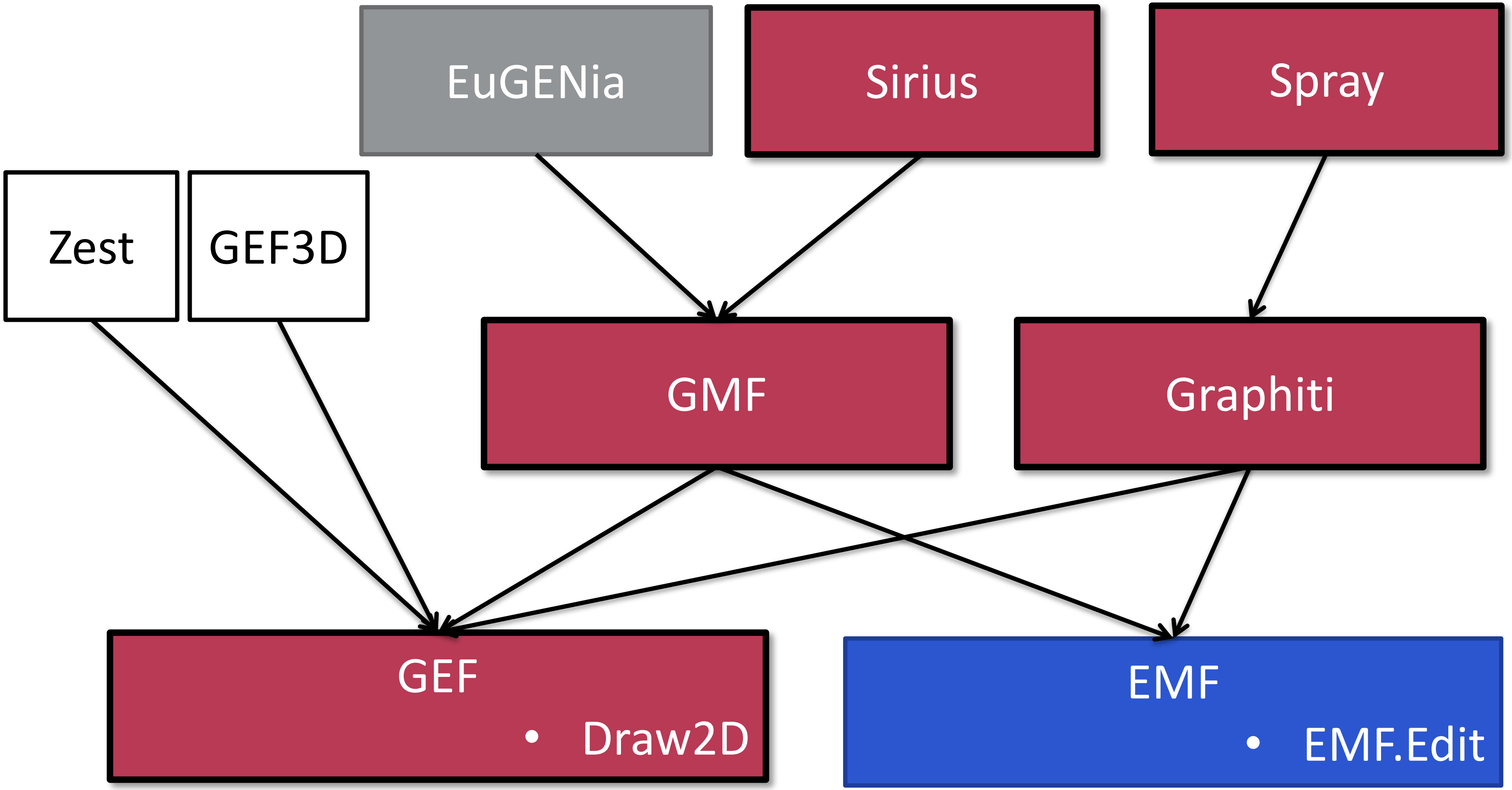
Language Server:

- CLI Eclipse bundled up in a .jar
- Xtext for textual DSL → EMF model
- ELK for auto-layouting diagram viewer

Graphical Editor Technologies in Eclipse

(supplementary material)

Graphical Editor Technologies



EuGENia

Sirius

Spray

Zest

GEF3D

GMF

Graphiti

GEF
- Draw2D

EMF
- EMF.Edit

Implementation

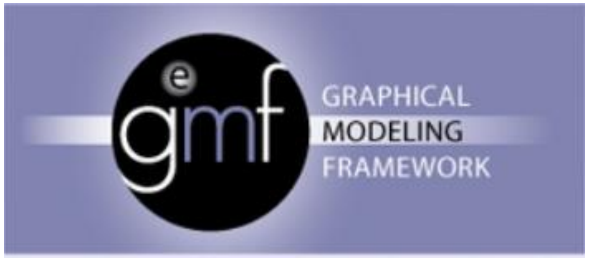
- Presentation
  - Based on a Canvas
  - Using vector-graphic libraries (GEF/Draw2d)
- Model manipulation
  - *EMF Edit* model manipulation commands
    - Atomic operations: create/modify/remove node/edge
  - Transactional modifications with *EMF Transactions*
    - Undo/redo support
- Notation/view model
  - Domain-independent implementation in GMF, Graphiti

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

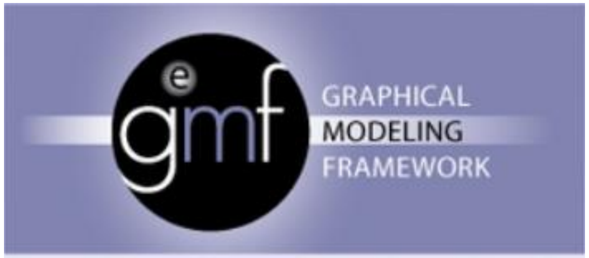
Technologies 2. – GMF

- 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

- (Relatively) 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
    - 1.0: 2014-06 (Kepler release train)
    - …
    - 6.3: 2019-06
    - …

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

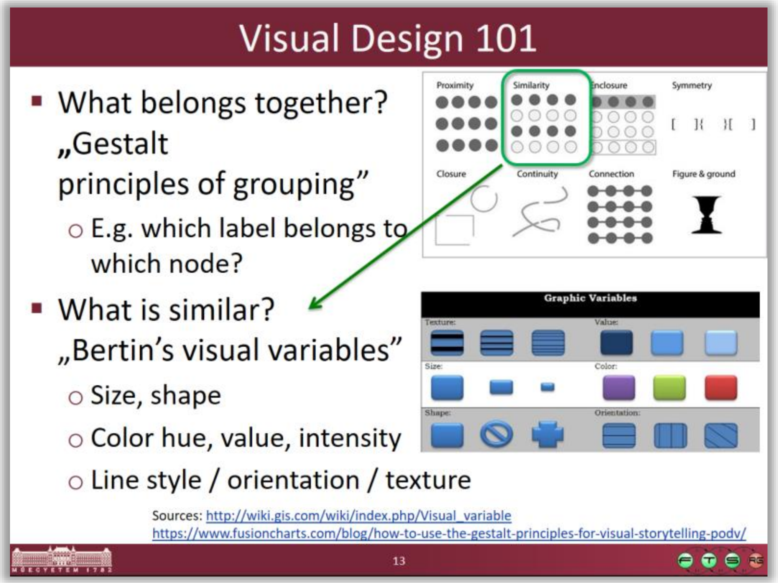
Concrete Syntax Design

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
  - Model(ing language)s can have multiple viewpoints
- Emerging standards for language servers

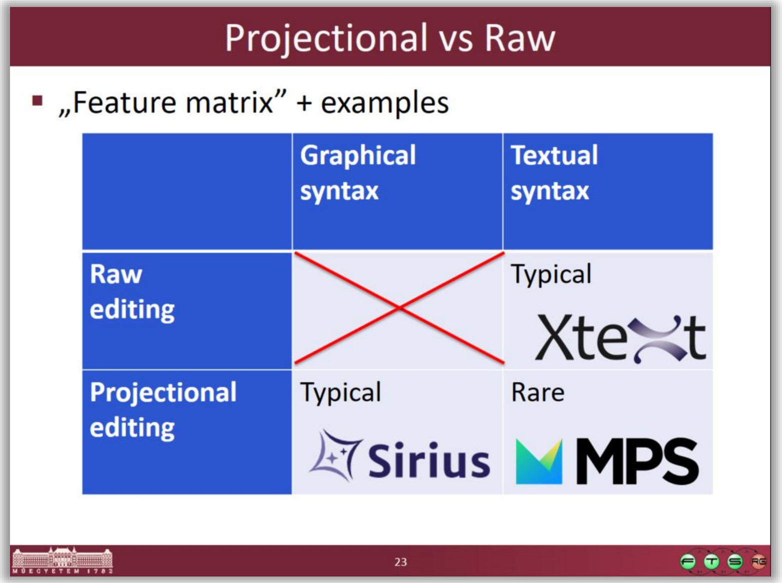
Summary



Visual Design 101

- What belongs together? „Gestalt principles of grouping"
  - E.g. which label belongs to which node?
- What is similar? „Bertin's visual variables"
  - Size, shape
  - Color hue, value, intensity
  - Line style / orientation / texture

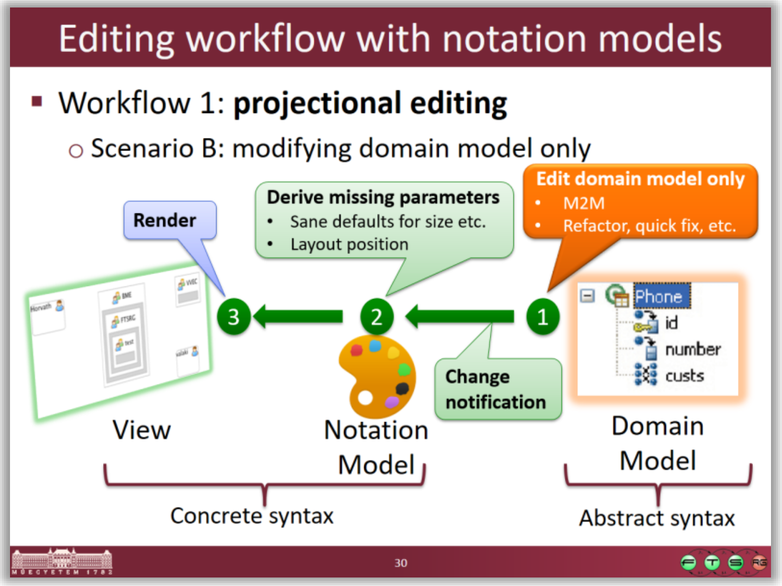
Sources: http://wiki.gis.com/wiki/index.php/Visual_variable
https://www.fusioncharts.com/blog/how-to-use-the-gestalt-principles-for-visual-storytelling-podv/



Projectional vs Raw

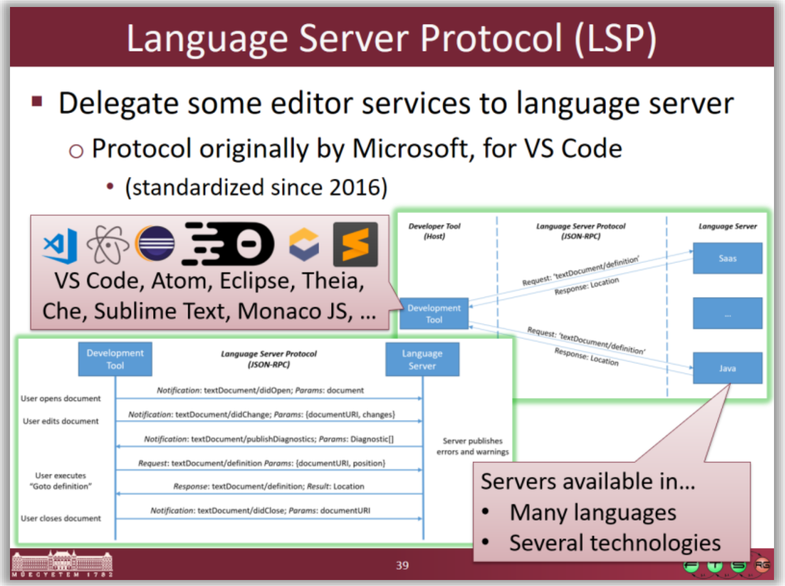
- „Feature matrix" + examples

	Graphical syntax	Textual syntax
Raw editing	✗	Typical Xtext
Projectional editing	Typical Sirius	Rare MPS



Editing workflow with notation models

- Workflow 1: **projectional editing**
  - Scenario B: modifying domain model only



Language Server Protocol (LSP)

- Delegate some editor services to language server
  - Protocol originally by Microsoft, for VS Code
    - (standardized since 2016)

VS Code, Atom, Eclipse, Theia, Che, Sublime Text, Monaco JS, …

Developer Tool (Host)
Language Server Protocol (JSON-RPC)
Language Server
Development Tool
Request: 'textDocument/definition'
Response: Location
Saas
…
Java
Development Tool
Language Server Protocol (JSON-RPC)
Language Server
User opens document
Notification: textDocument/didOpen; Params: document
User edits document
Notification: textDocument/didChange; Params: {documentURI, changes}
Notification: textDocument/publishDiagnostics; Params: Diagnostic[]
Server publishes errors and warnings
Request: textDocument/definition Params: {documentURI, position}
User executes "Goto definition"
Response: textDocument/definition; Result: Location
User closes document
Notification: textDocument/didClose; Params: documentURI

Servers available in…
- Many languages
- Several technologies