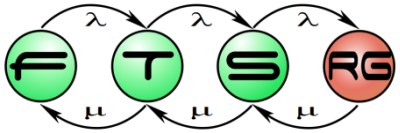
Concrete Syntax Design for Domain-specific Languages

Zoltán Ujhelyi

Model Driven Software Development

Lecture 7



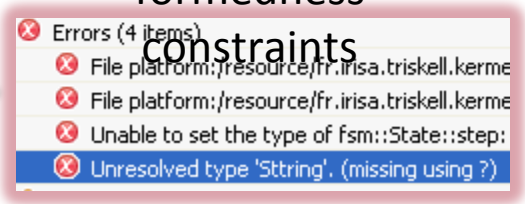
Budapesti Műszaki és Gazdaságtudományi Egyetem
Méréstechnika és Információs Rendszerek Tanszék

Structure of DSMs

Graphical syntax

Abstract syntax

Well-formedness constraints

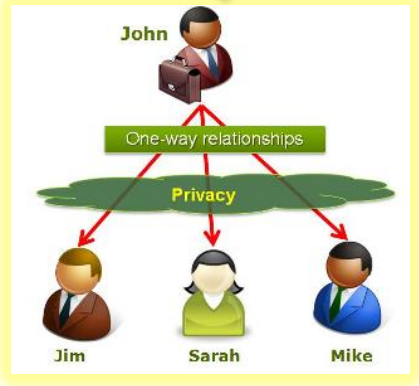


Behavioural semantics, simulation

Code generation

Mapping

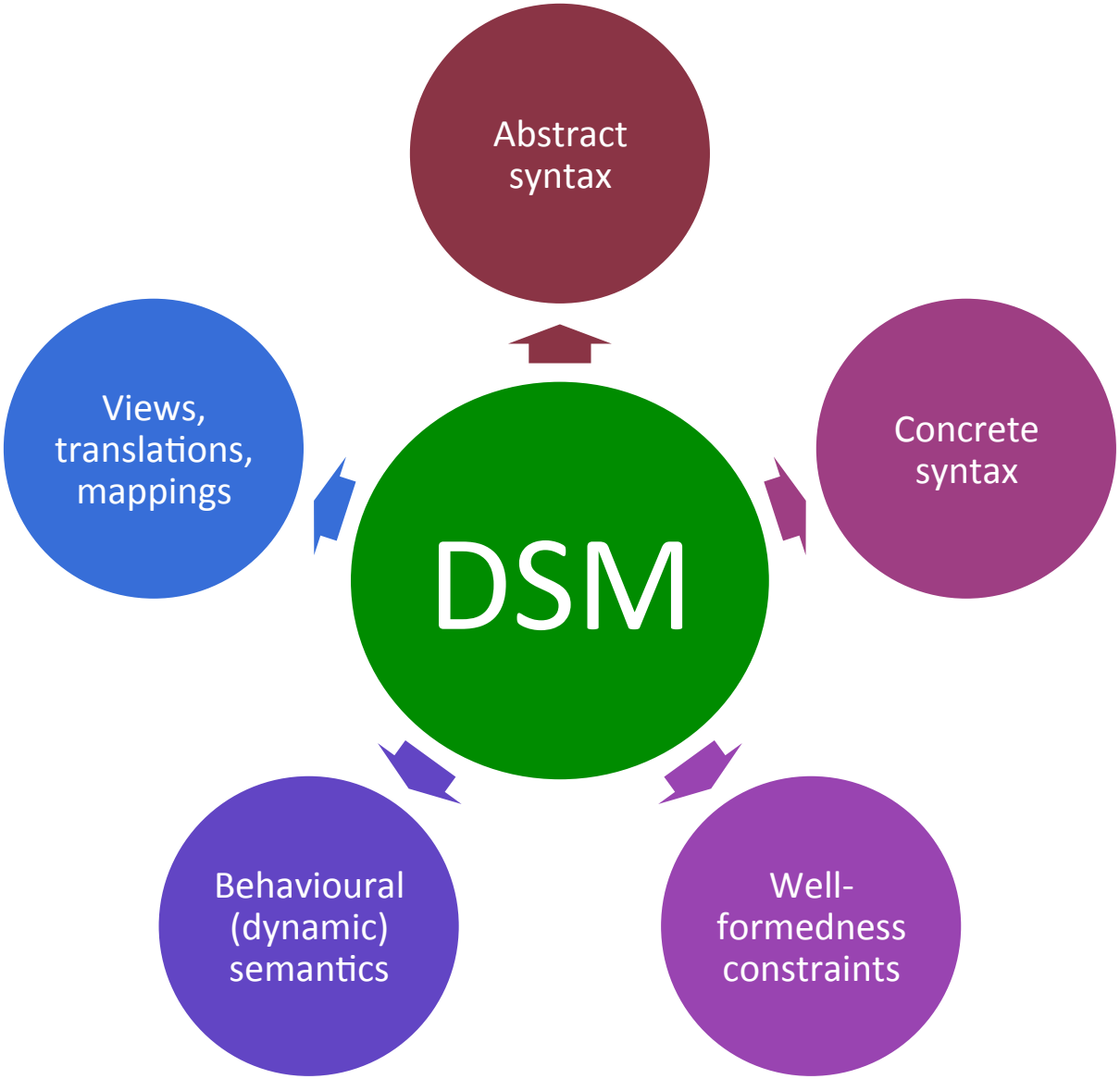
Textual syntax



View

Code (documentation, configuration)

DSM aspects


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

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
  - 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

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 GPL

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 GPL

Technology

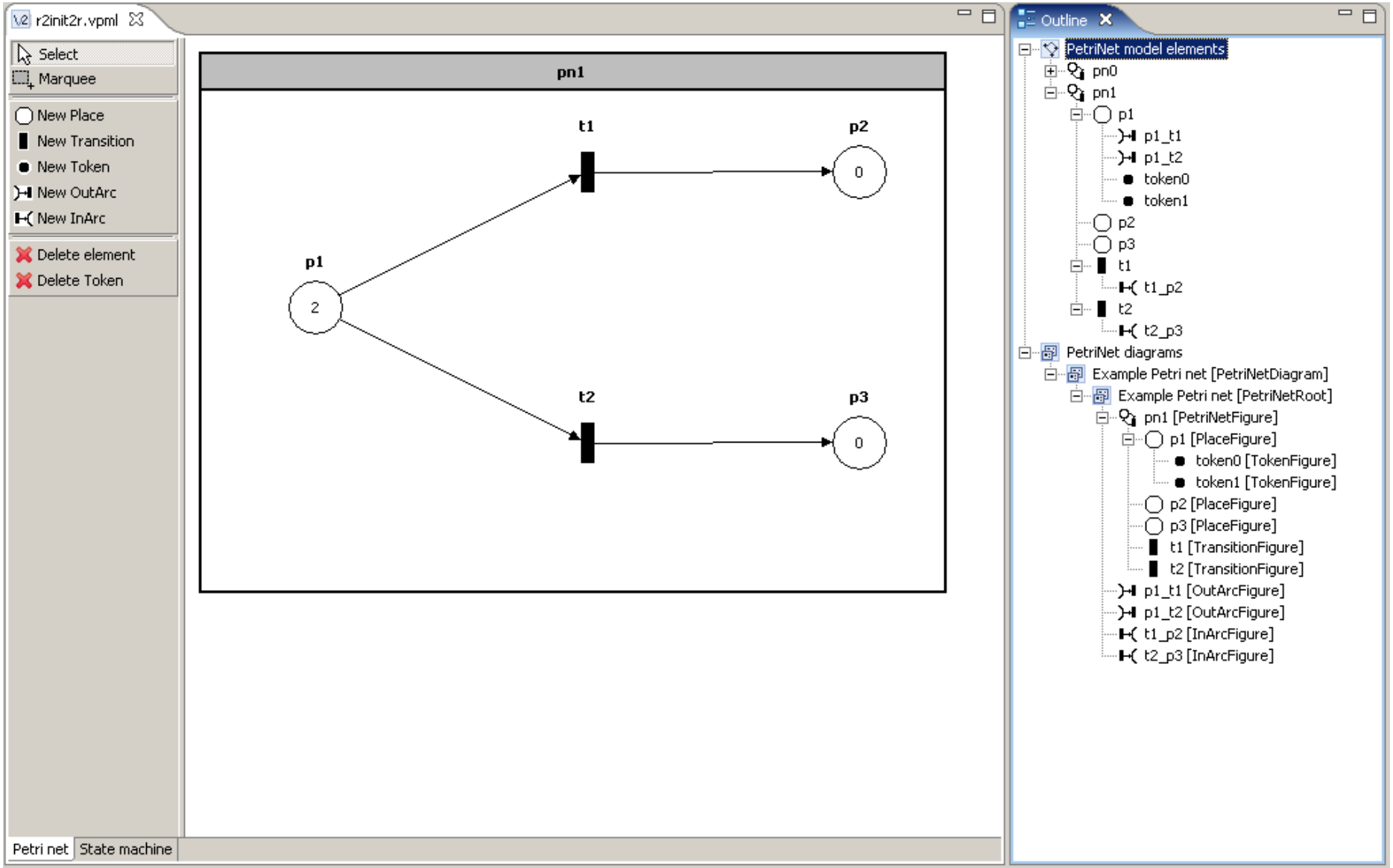
- Eclipse Modeling Tools
  - Several related subprojects
  - Each supports a single aspect
  - Examples of today
- Microsoft Visual Studio 2010 Visualization & Modeling SDK
  - DSL modeling framework from Microsoft
  - Own metamodeling core
  - Focuses on graphical modeling

Graphical Editors

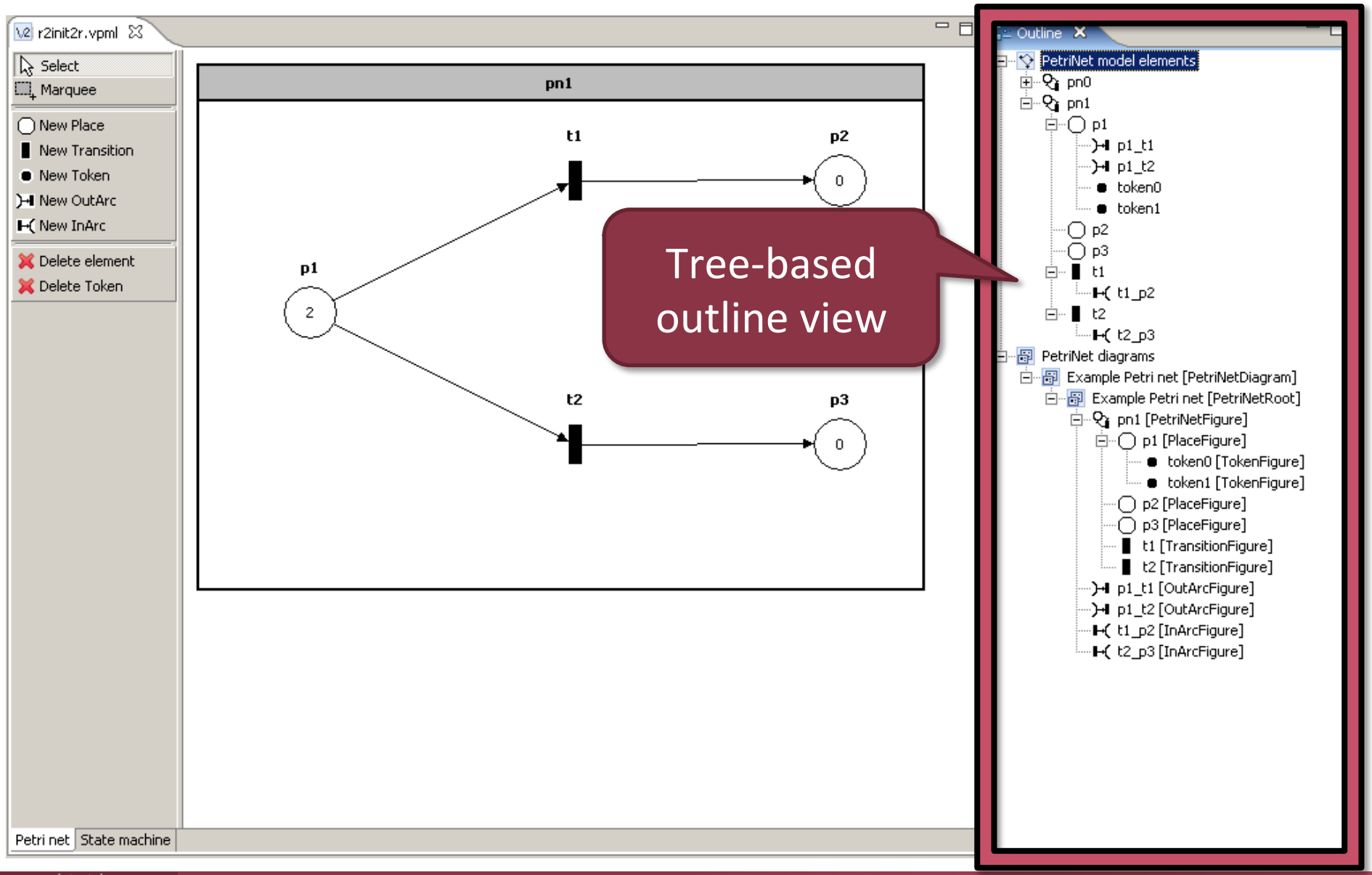
Graphical Modeling

- Model
  - Typically graph-based modeling
  - In our case EMF
- Idea
  - Display and editing as a graph model

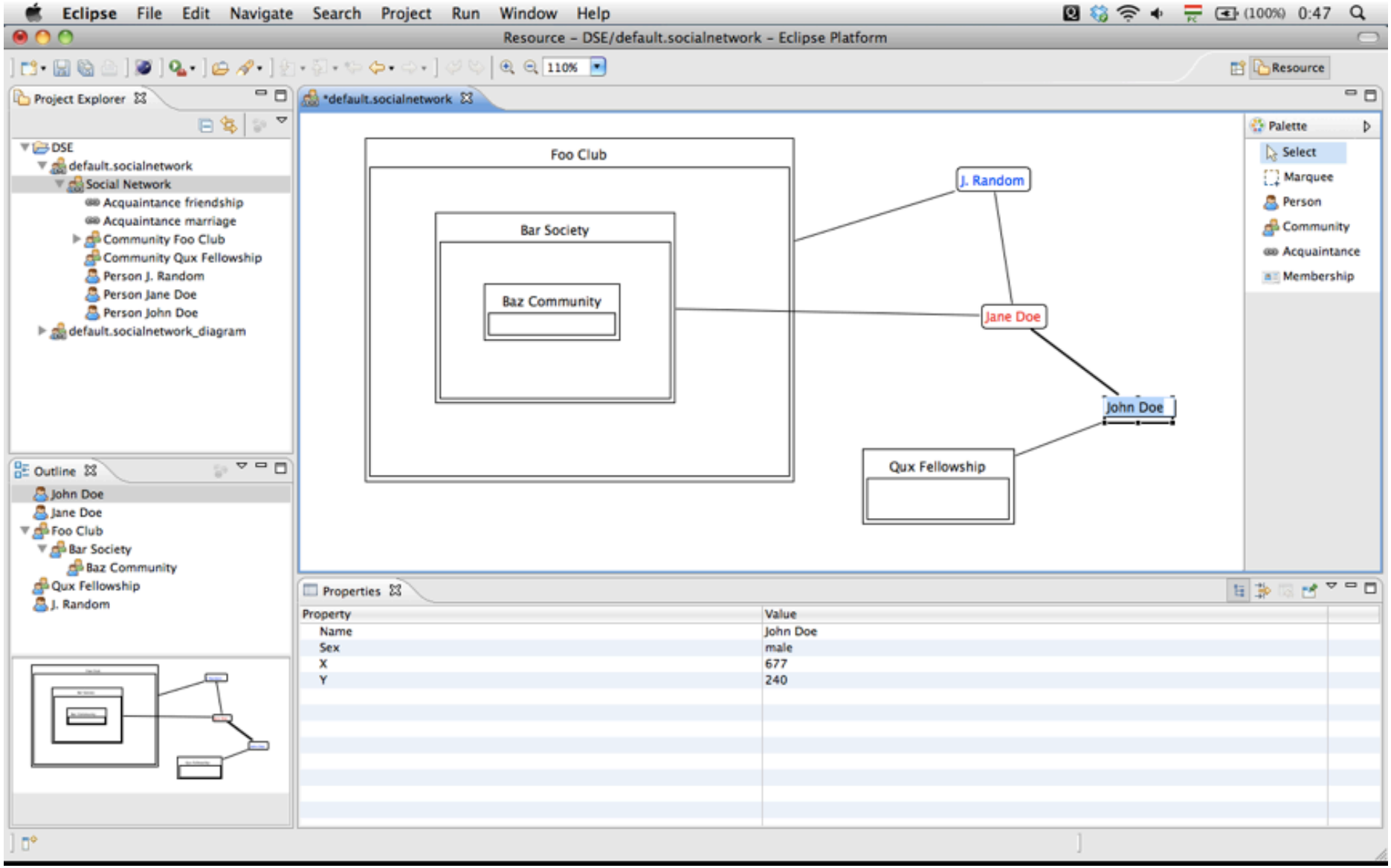
Example: Petri net editor



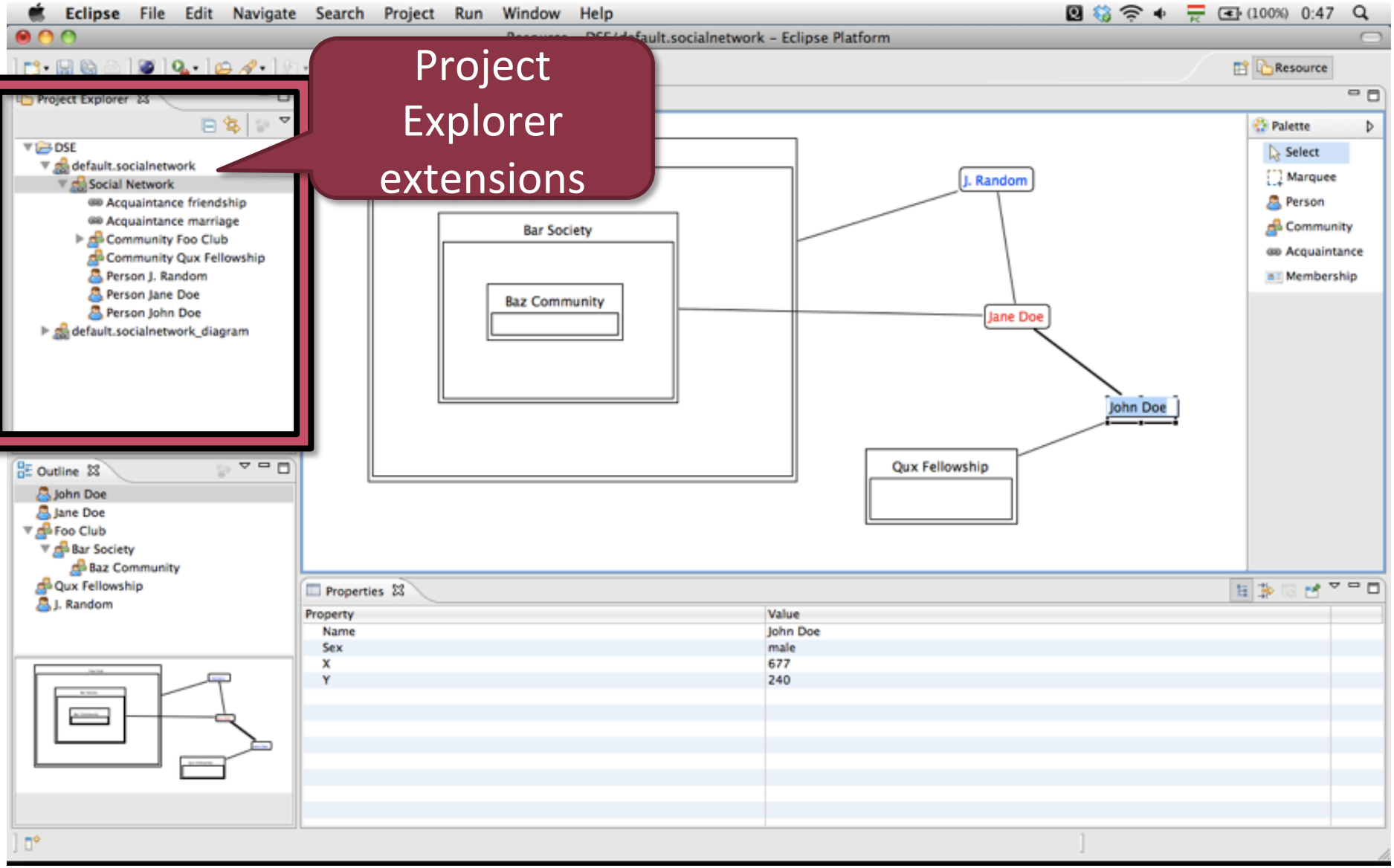
Example: Petri net editor



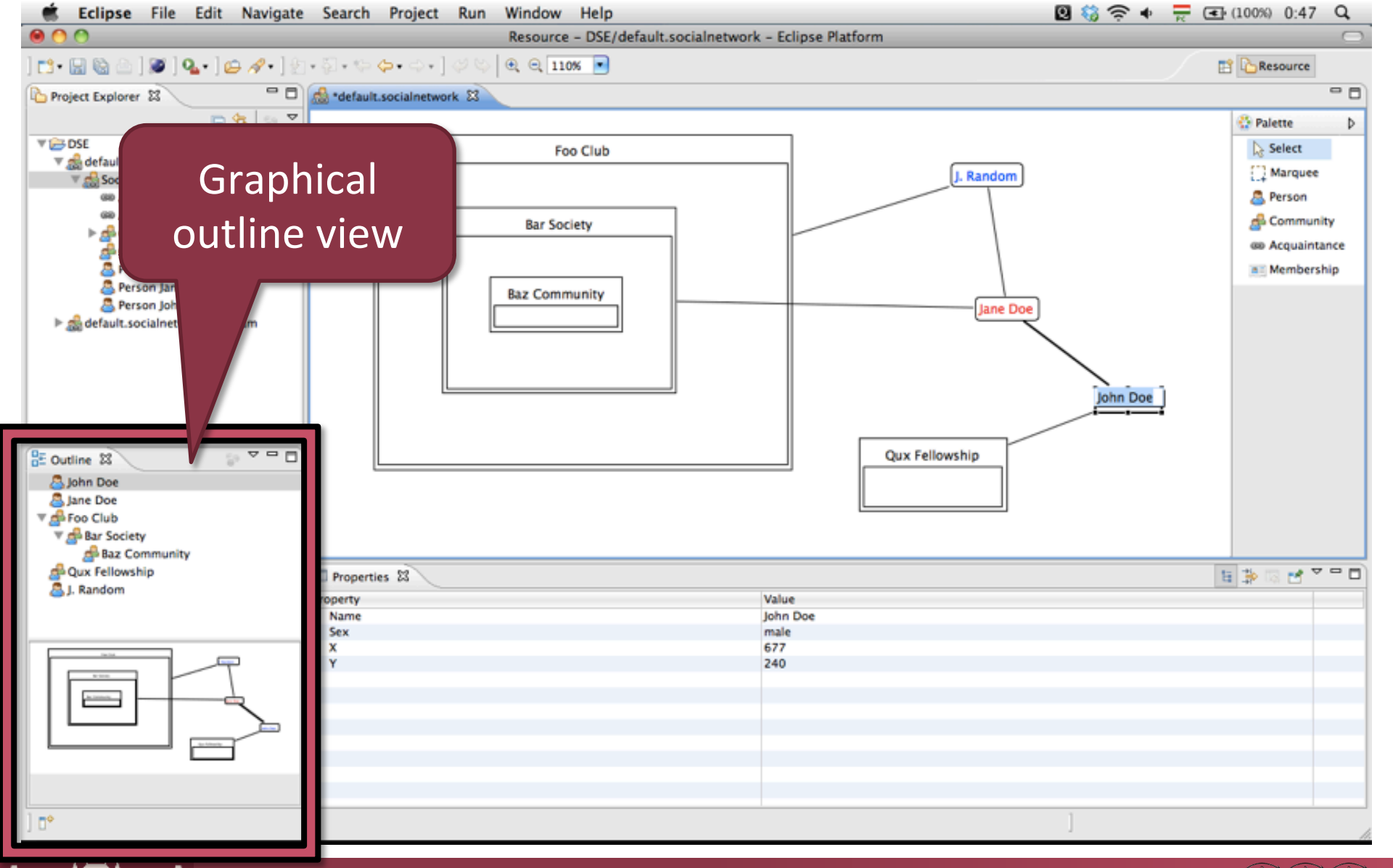
Example: Social Network editor



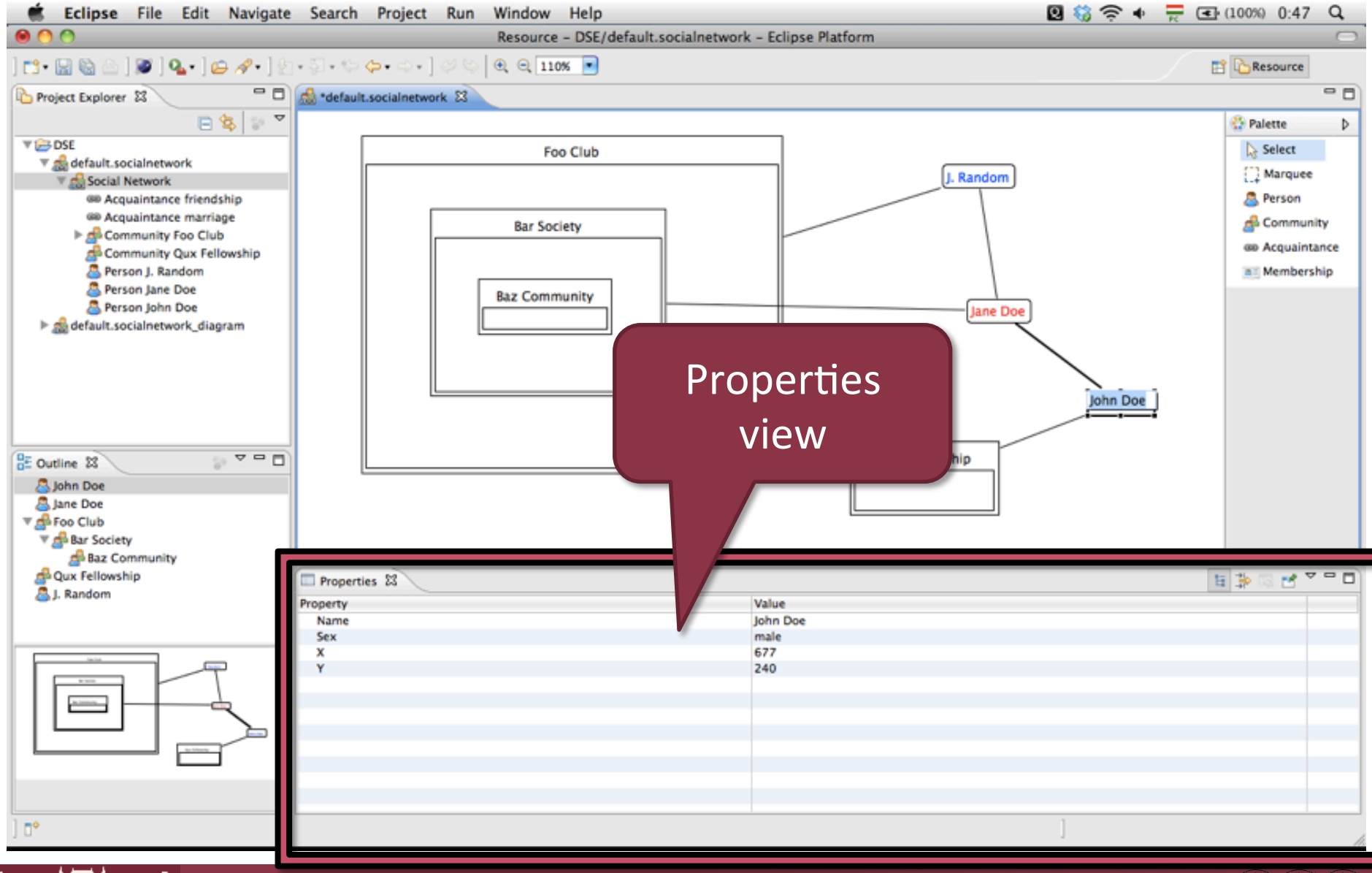
Example: Social Network editor



Example: Social Network 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

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

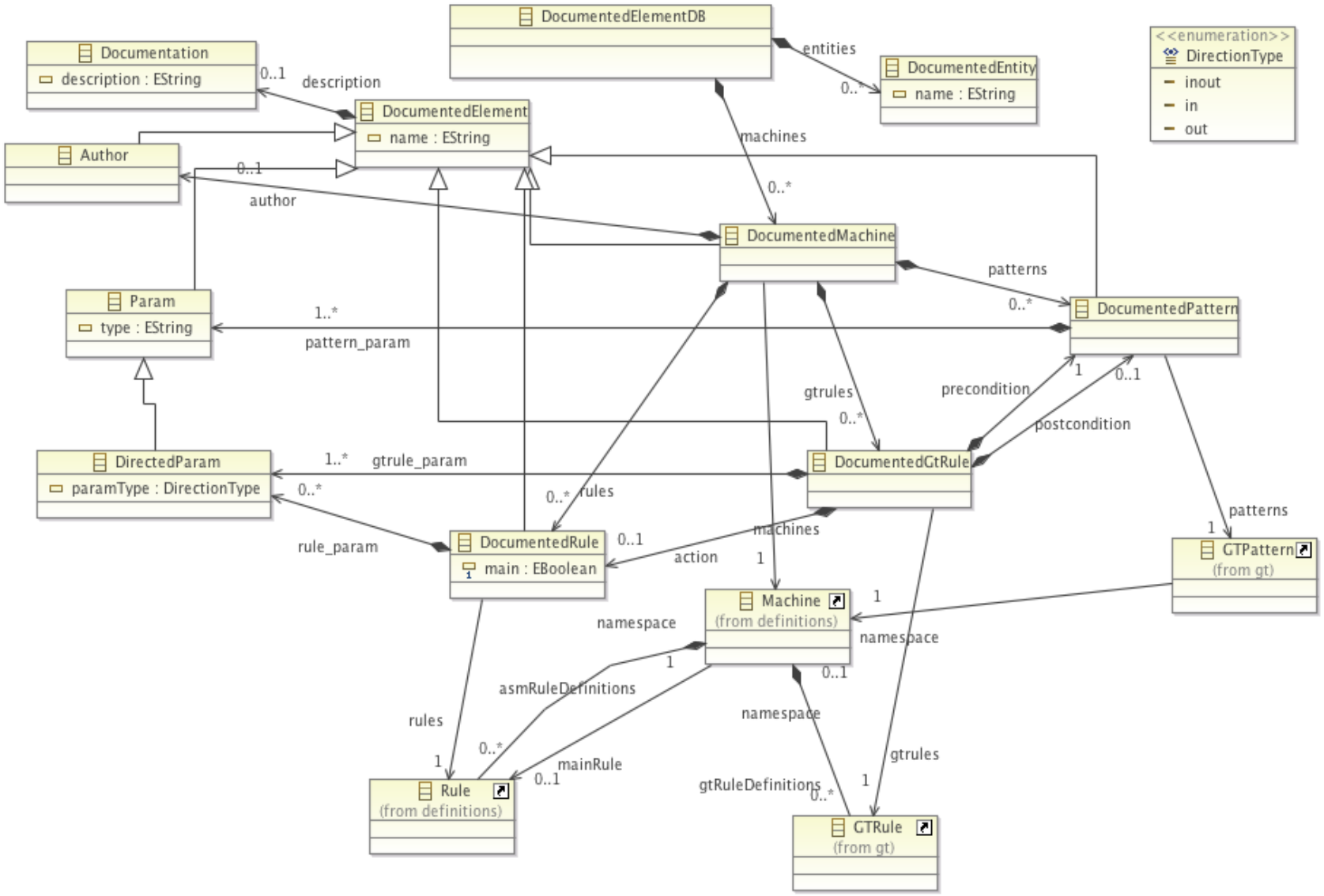
Technology Comparison

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

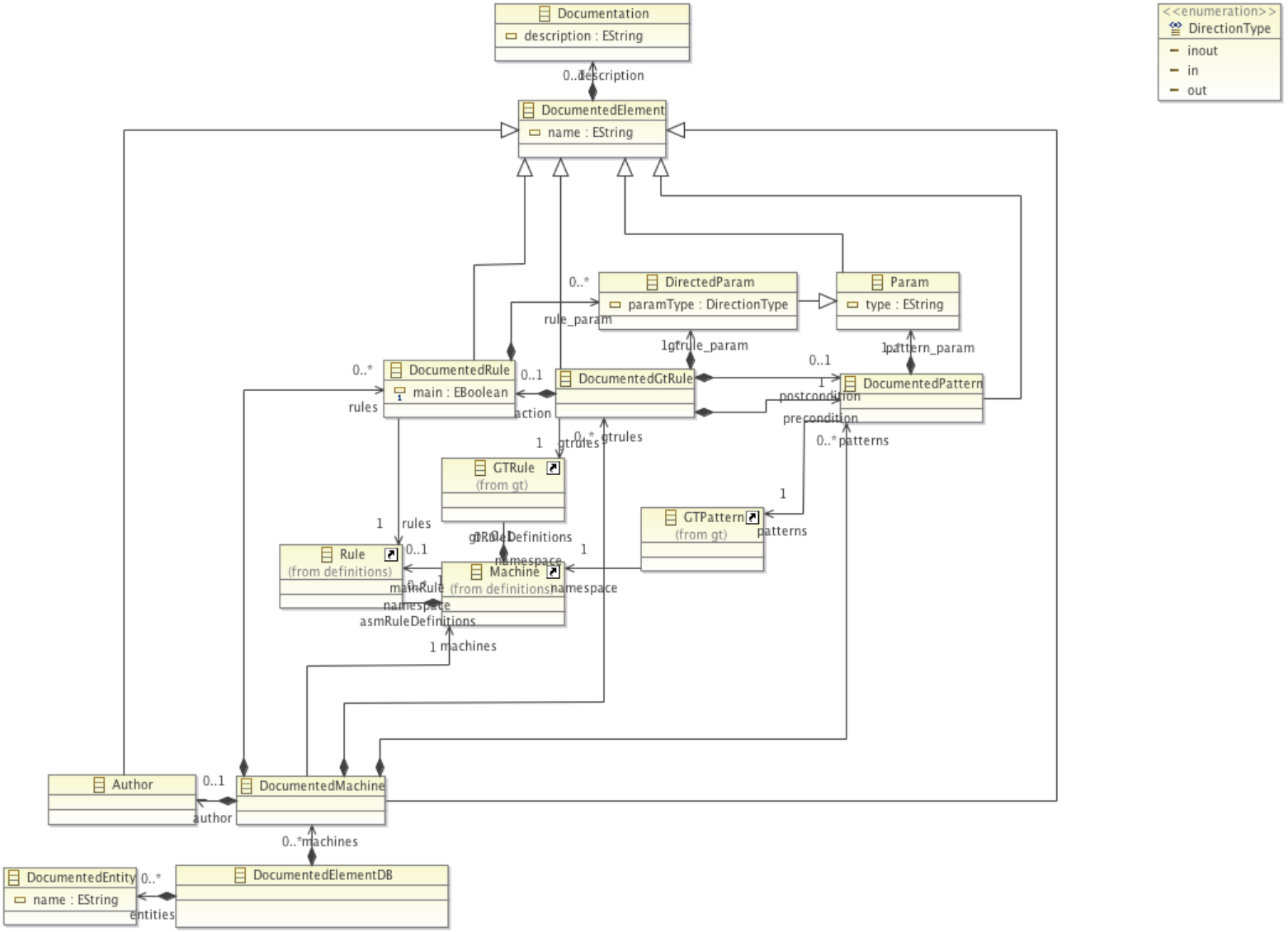
Advanced issues

- Cumbersome editing
  - E.g., reorganization to instert 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

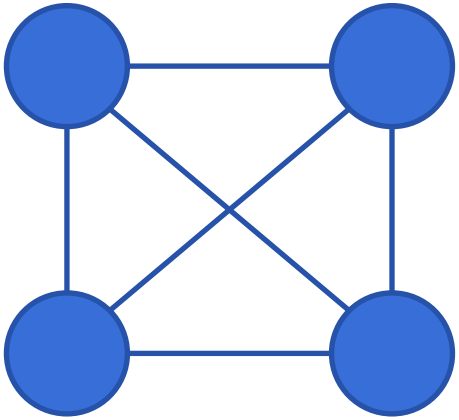


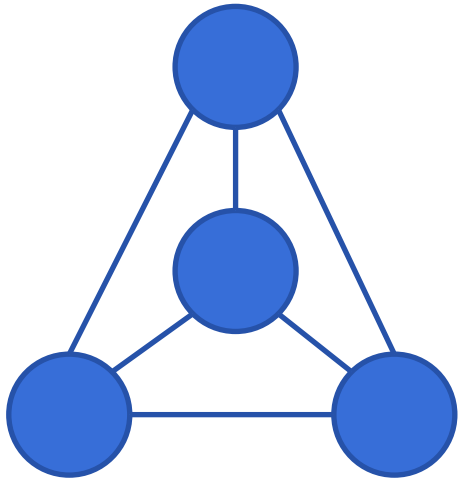
Example: Layouting



Layouting Support for Graphical Editors

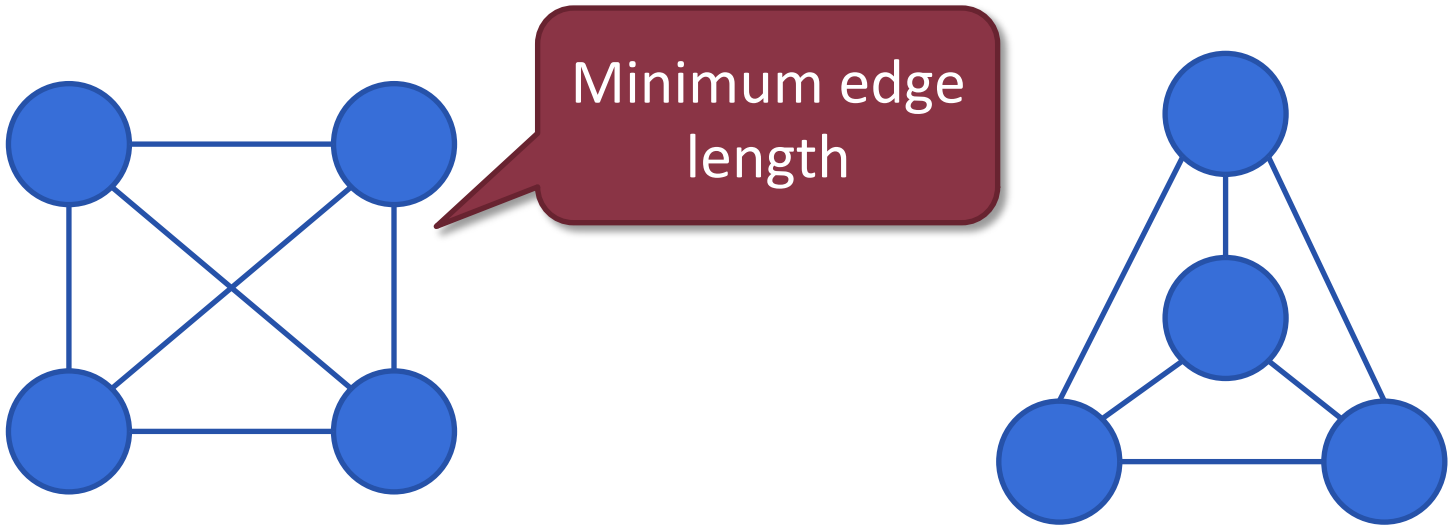
- Computation of the position of nodes
  - Possible to do automatically
  - For a given metamodel
    - No unified visual requirements possible
    - We have to decide what is important to show





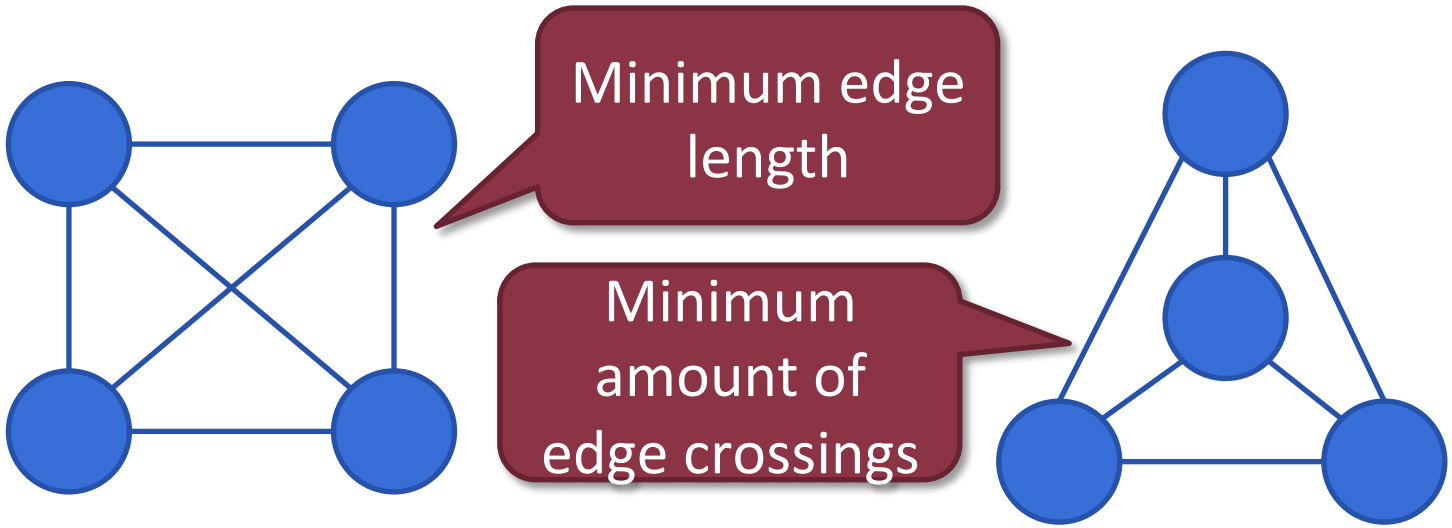
Layouting Support for Graphical Editors

- 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

- 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
  - Graph widgets for SWT applications
  - Easily integratable into Eclipse applications
  - Not the best layout algorithms
- **KIELER** - http://rtsys.informatik.uni-kiel.de/trac/kieler/
  - Eclipse based tools
  - Built-in support for GMF layouting

Textual modeling languages

Textual Domain-specific Languages

- Idea
  - Describing models as text files
- Textual development
  - Long history (30+ years)
  - Well-researched theory
  - Mature tools

Regular expressions

- Pattern matching for strings
  - Good support
    - Most programming languages
    - More or less the same syntax
  - Calculates and returns matches
- Usable as DSL parser?

RegExp: Validation of email addresses

(?:(?:\r\n)?[\t])*(?:(?:(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[\t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[\t]))*"(?:(?: \r\n)?[\t])*)(?:\.(?:(?:\r\n)?[\t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[\t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[\t]))*"(?:(?:\r\n)?[\t])*))*@(?:(?:\r\n)?[\t])*(?:[^()<>@,;:\\".\[\] \000-\0 31]+(?:(?:(?:\r\n)?[\t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[\t])*)(?:\.(?:(?:\r\n)?[\t])*(?:[^()<>@,;:\\".\[\] \000-\031]+ (?:(?:(?:\r\n)?[\t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?: (?:\r\n)?[\t])*))*|(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[\t])+|\Z |(?=[\["()<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[\t]))*"(?:(?:\r\n)?[\t])*)(?:\.(?:(?:\r\n)?[\t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\ r\n)?[\t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[\t])*)(?:\.(?:(?:\r\n)?[\t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[\t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[\t])*))*) *:(?:(?:\r\n)?[\t])*)?(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[\t])+ |\Z|(?=[\["()<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[\t]))*"(?:(?:\r \n)?[\t])*)(?:\.(?:(?:\r\n)?[\t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?: \r\n)?[\t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[\t]))*"(?:(?:\r\n)?[\t])*)*@(?:(?:\r\n)?[\t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[\t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[\t])*)(?:\.(?:(?:\r\n)?[\t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(? :(?:(?:\r\n)?[\t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(? :(?:\r\n)?[\t])*))*\>(?:(?:\r\n)?[\t])*)|(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[\t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)? [\t]))*"(?:(?:\r\n)?[\t])*)(?:\.(?:(?:\r\n)?[\t])*(?:[^()<> @,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[\t])+|\Z|(?=[\["()<>@,;:\\" .\[\]]))|" (?:[^\"\r\\]|\\.|(?:(?:\r \n)?[\t]))*"(?:(?:\r\n)?[\t])*))*@(?:(?:\r\n)?[\t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[\t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[\t])*)*|(?:[^()<>@,;:\\".\[\] \000- \031]+(?:(?:(?:\r\n)?[\t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[\t]))*"(?:(?:\r\n)?[\t])*)*\<(?:(?:\r\n)?[\t])*(?:@(?:[^()<>@,; :\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[\t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[\t])*)(?:\.(?:(?:\r\n)?[\t])*(?:[^()<>@,;:\\" .\[\] \000-\031]+(?:(?:(?:\r\n)?[\t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]]\r\\]|\\.)*\](?:(?:\r\n)?[\t])*))*(?:,@(?:(?:\r\n)?[\t])*(?:[^()<>@,;:\\" .\[\] \000-\031]+(?:(?:(?:\r\n)?[\t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\] \r\\]|\\.)*\](?:(?:\r\n)?[\t])*)(?:\.(?:(?:\r\n)?[\t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[\t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\] \r\\] |\\.)*\](?:(?:\r\n)?[\t])*))*:(?:(?:\r\n)?[\t])*)?(?:[^()<>@,;:\\".\[\] \0 00-\031]+(?:(?:(?:\r\n)?[\t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|\\ .|(?:(?:\r\n)? [\t]))*"(?:(?:\r\n)?[\t])*)(?:\.(?:(?:\r\n)?[\t])*(?:[^()<>@, ;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[\t])+|\Z|(?=[\["()<>@,;:\\". \[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[\t])*(?:\.(?:(?:\r \n)?[\t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[\t])+|\Z|(?=[\["()<>@,;:\\" ".\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[\t]))*"(?:(?:\r\n)?[\t])*)*@(?:(?:\r\n)?[\t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[\t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[\t])*)(?:\.(?:(?:\r\n)?[\t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[\t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[\t])*))*\>(?:(?:\r\n)?[\t])*)(?:,\s*(?:(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[\t])+|\Z|(?=[\["()<>@,;:\\" ".\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[\t]))*"(?:(?:\r\n)?[\t])*)(?:\.(?:(?:\r\n)?[\t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[\t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[\t]))*"(?:(?:\r\n)?[\t])*)*@(?:(?:\r\n)?[\t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[\t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[\t])*)(? :\.(?:(?:\r\n)?[\t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[\t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[\t])*))*|(?: [^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[\t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[\t]))*"(?:(?:\r\n)?[\t])*)*\<(?:(?:\r\n) ?[\t])*(?:@(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[\t])+|\Z|(?=[\[" ()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[\t])*)(?:\.(?:(?:\r\n) ?[\t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[\t])+|\Z|(?=[\["()<> @,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[\t])*))*(?:,@(?:(?:\r\n)?[\t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[\t])+|\Z|(?=[\["()<>@, ;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[\t])*)(?:\.(?:(?:\r\n)?[\t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[\t])+|\Z|(?=[\["()<>@,;:\\". \[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[\t])*(?:\.(?:(?:\r \n)?[\t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[\t])+|\Z|(?=[\["()<>@,;:\\" ".\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[\t]))*"(?:(?:\r\n)?[\t])*)*@(?:(?:\r\n)?[\t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[\t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[\t])*)(?:\.(?:(?:\r\n)?[\t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[\t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[\t])*))*\>(?:(?: ?:\r\n)?[\t])*))?;\s*)

This regular expression will only validate addresses that have had any comments stripped and replaced with whitespace

Additionally

- Output is a single boolean variable
  - Decides whether the string matches the language
- What is missing?

Additionally

- Output is a single boolean variable
  - Decides whether the string matches the language
- What is missing?

Error localization!

Example: Grammar for describing name lists

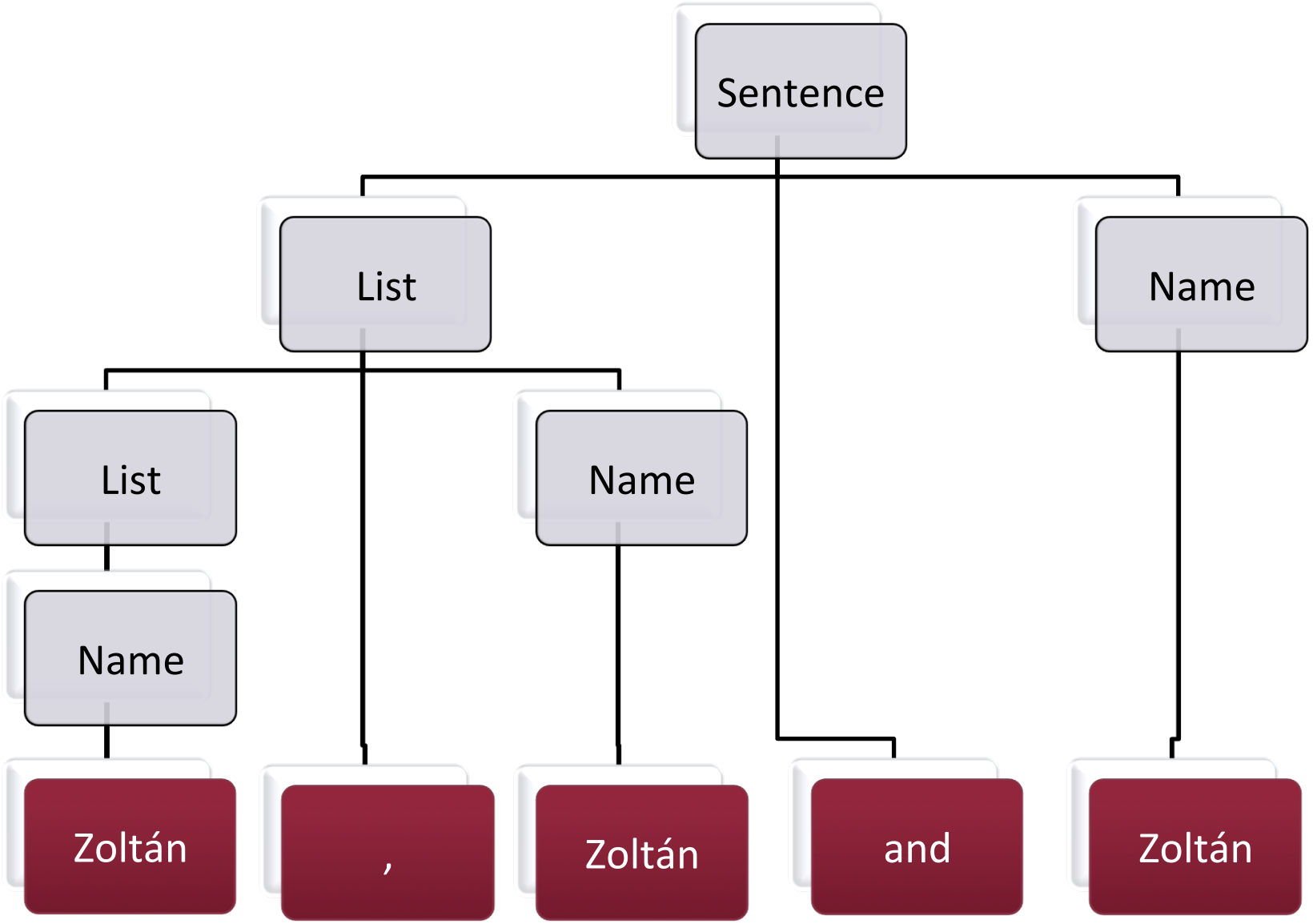
- Terminal Symbols
  - *“Dániel”*, *“István”*, *“Zoltán”*, *“and”*, *“,”*
- Non-terminal Symbols
  - «Name», «Sentence», «List»

«Name»::= *Zoltán* | *István* | *Dániel*

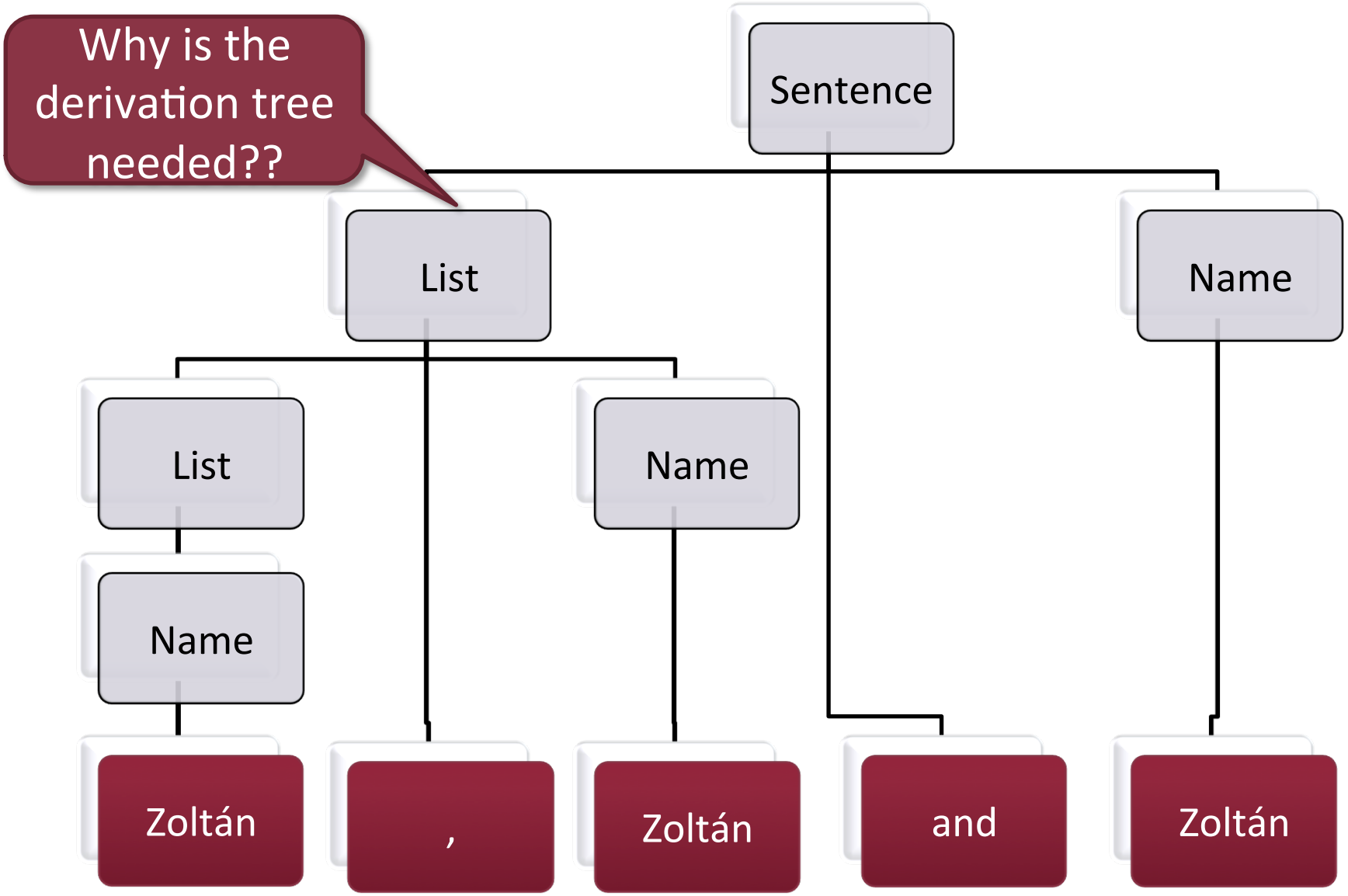
«Sentence»::= «Name» | «List» *and* «Name»

«List»::= «List», «Name»| «Name»

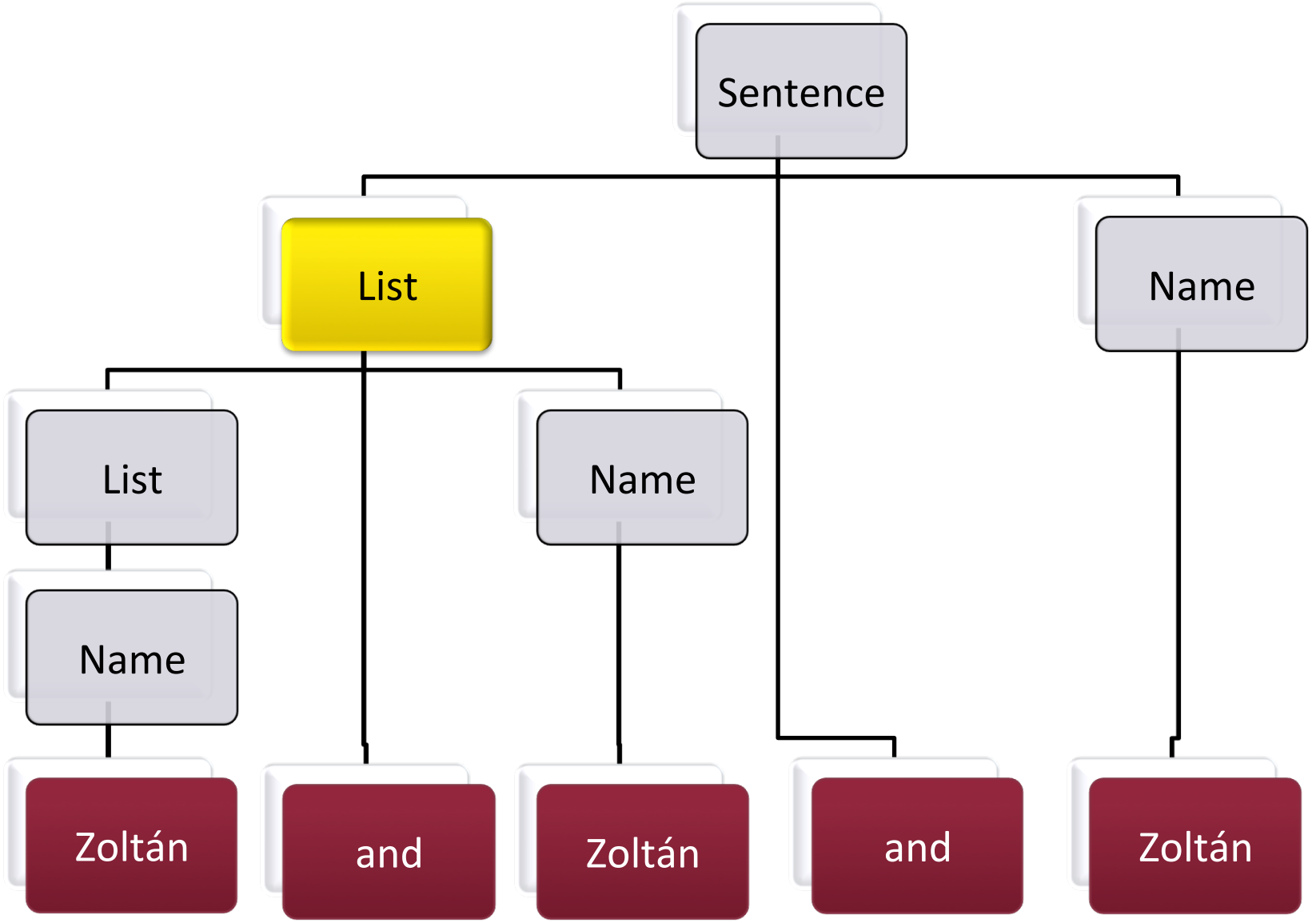
Derivation Tree



Derivation Tree



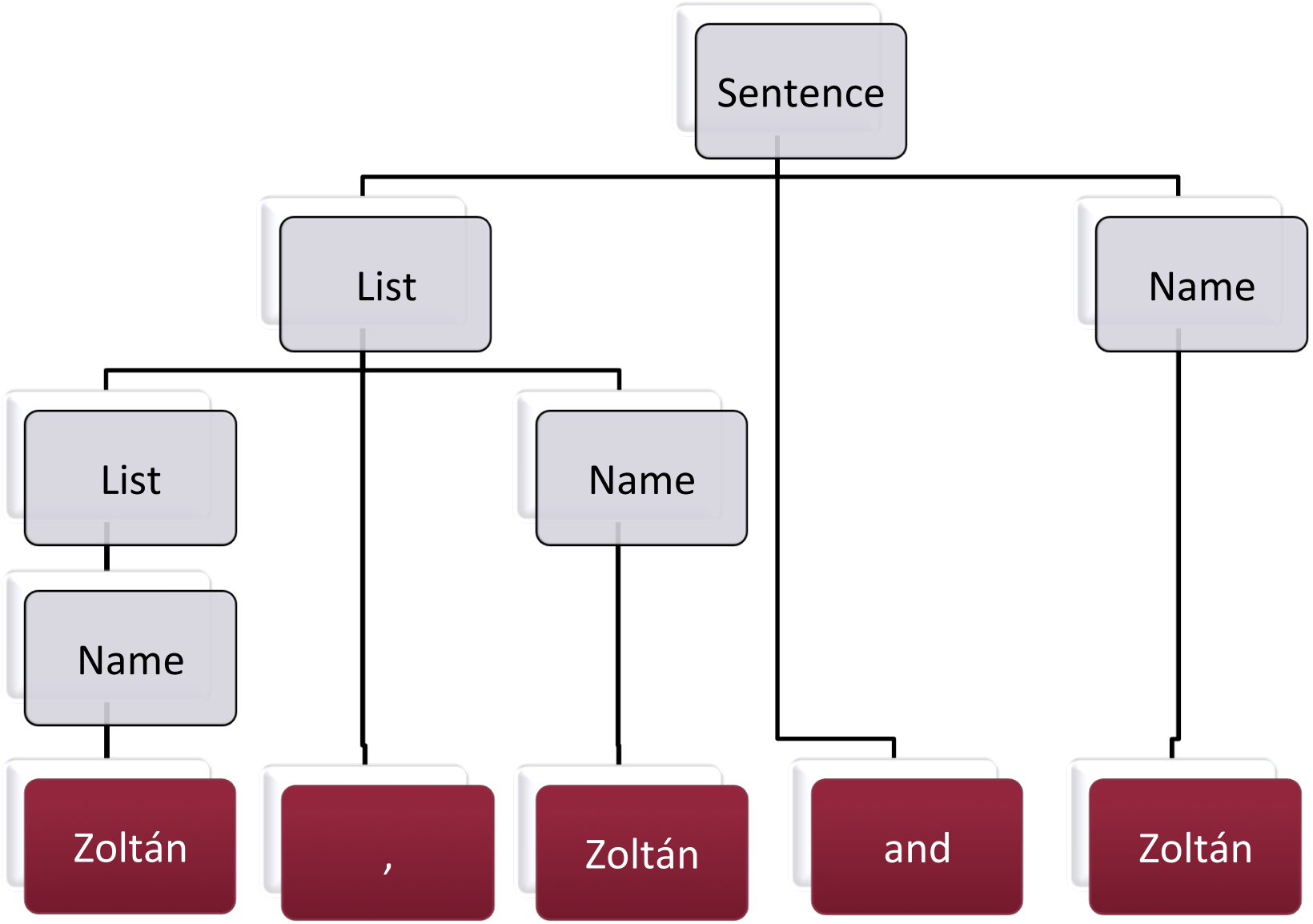
Derivation Tree



Additional practical tasks

- Handling input strings
- Variable handling
- High level analysis

Input



Input

Input string:

'Z' 'o' 'l' 't' 'á' 'n' ' ' ' ' ',' ' 'Z'
'o' 'l' 't' 'á' 'n' ' ' ' 'a' 'n' 'd'
' ' 'Z' 'o' 'l' 't' 'á' 'n'

Zoltán | , | Zoltán | and | Zoltán

Input handling

Input:

Character stream

Parser input:

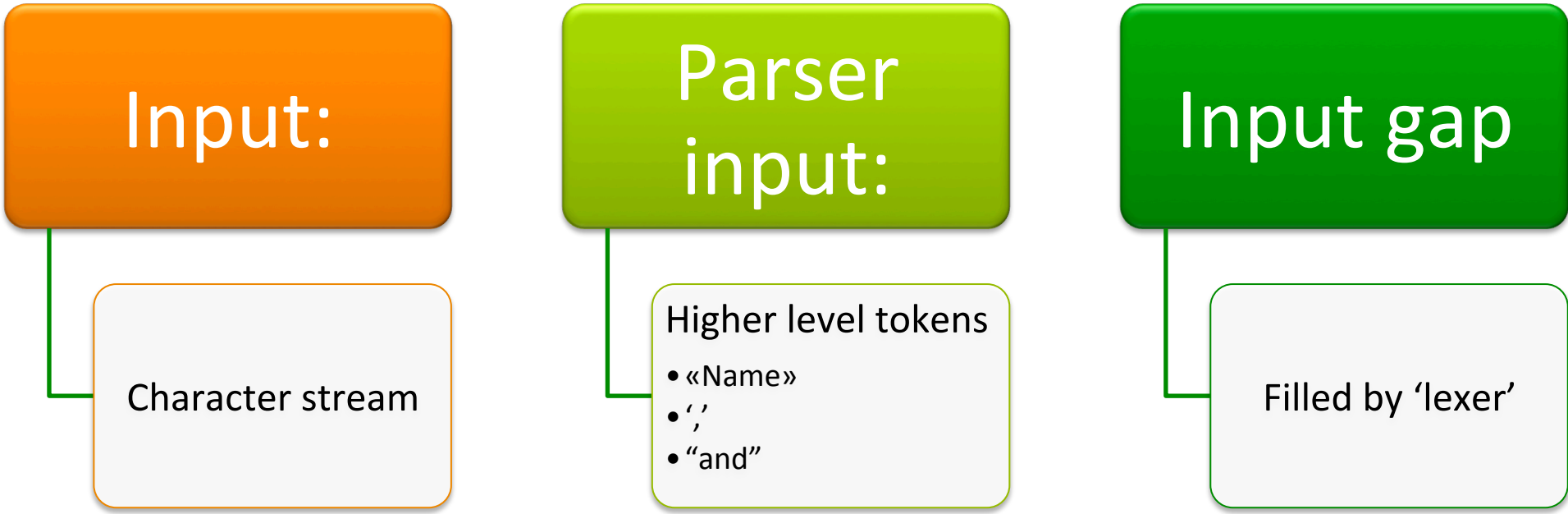
Higher level tokens

- «Name»
- ‘,’
- “and”

Input gap

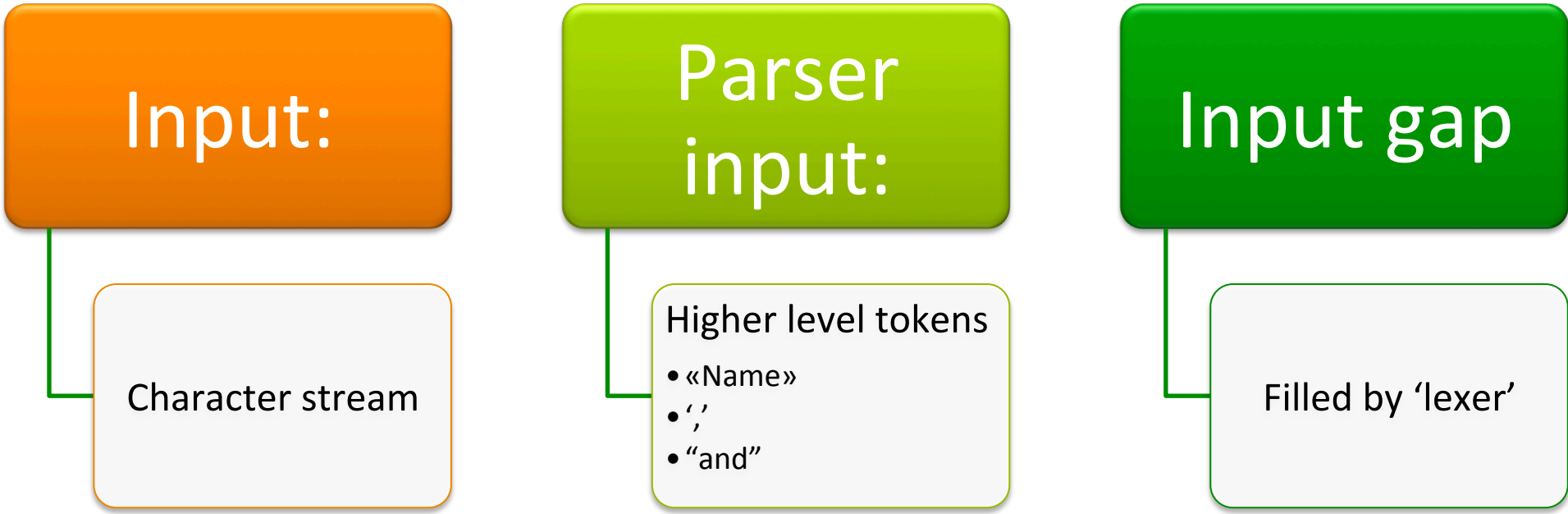
Filled by ‘lexer’

Input handling



- Why is this indirection useful?
  - Error handling
  - Performance
  - Problem decomposition

Input handling



- Why is this indirection useful?
  - Error handling
  - Performance
  - Problem decomposition

Lexer

- Goal:
  - Tokenizing the input character stream
  - Similar to the parsing problem
    - But usually simpler – Typically regular expressions
    - Only word/token identification
    - Optional task: leaving out comments
  - Simplifies parsing significantly

Variable Handling

```
a=3;
System.out.println(a);
```

- Variables
  - In runtime
    - Value calculation/substitution
  - Editing/analysis time
    - References to other parts of the AST

Variable Handling

- *Variable reference*
  - A use of an already defined variable
- *Variable definition*
  - A declaration of a variable
  - Unique naming
    - Variable definitions must be resolvable
    - Extra phase required after parsing

Variable Handling

- Parser checks
  - “Can a variable named like ‘a’?”
- Reference resolution
  - “Is a variable defined?”
    - Scoping problem
  - “Is a variable uniquely defined?”

Scoping Problem

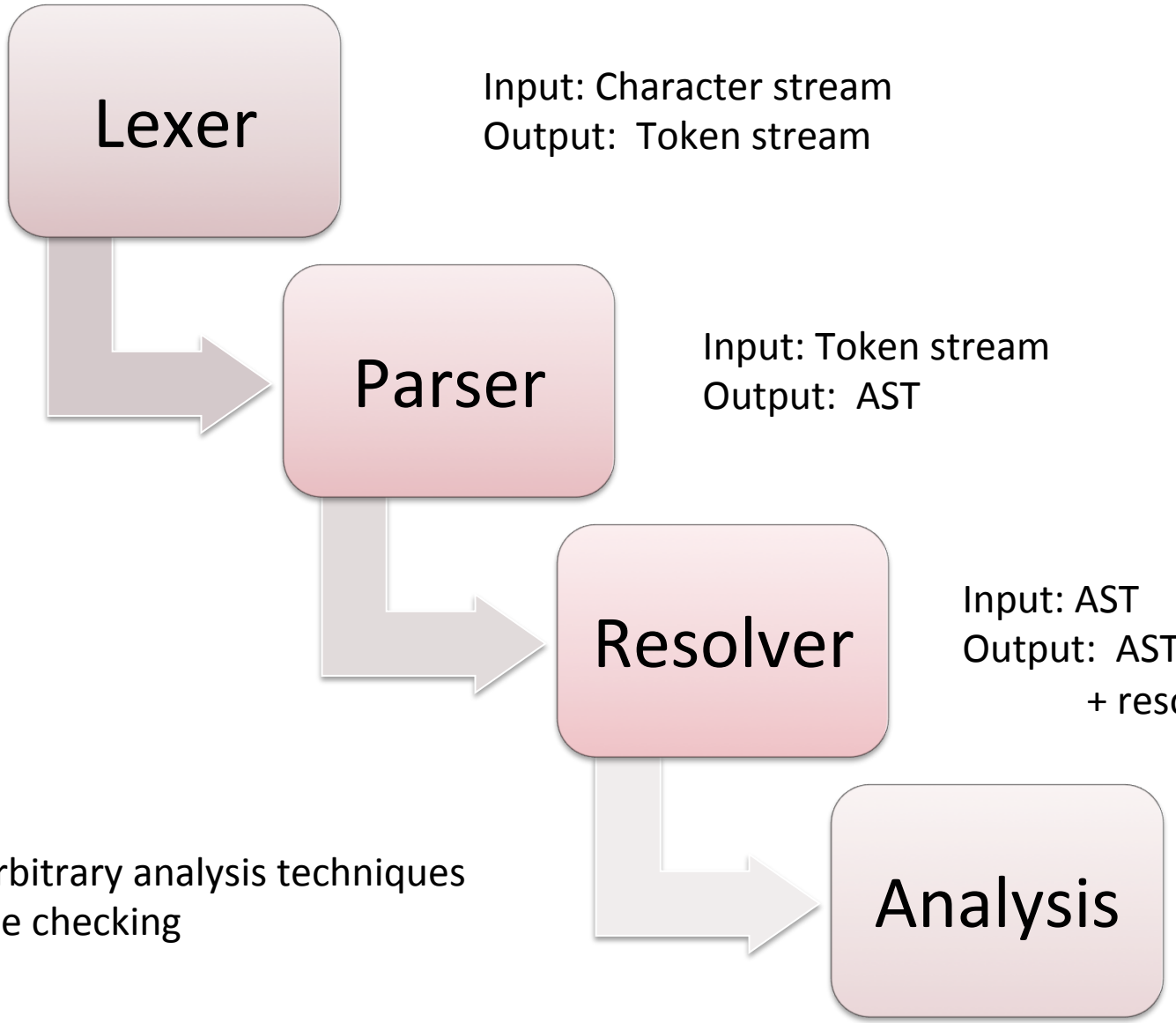
```
private int value;

public void setValue(int value) {
    this.value = value;
}
```

Which variable declaration is referred by ‘value’?

Scoping Problem

- Solution
  - Resolver should define this
- Possible approaches
  - Most specific declaration
  - Conflict is error
  - Qualified references
  - …

The Parsing Process



Technologies 1. – IMP

- IDE Meta-tooling platform
  - Goal:
    - Language editor creation
  - Every parser-generator should be re-usable
  - Manual coding required
  - Project is close to dead

Technologies 2. - EMFText

- Editor generation
  - Based on existing EMF model
  - Different generated grammar styles
    - Manually modifiable
    - Limited grammar support
- Syntax Zoo
  - ~100 different syntax examples available
  - Including a Java implementation (called JaMoPP)

Technologies 3. – Xtext

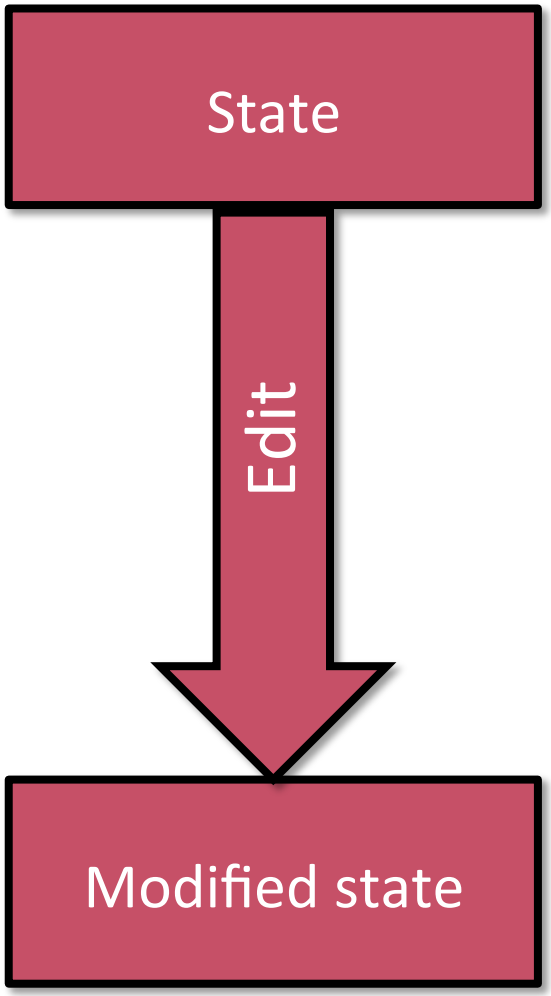
- Editor and EMF model generation
  - Based on grammar
  - Optionally can be initiated from an existing EMF model
- Easy to work with
  - But highly customizable
  - Both the grammar and the generated code
- New development – DSLs over JVM:
  - Xbase expression language
  - JVM Model Inference instead of code generation

Textual or graphical?

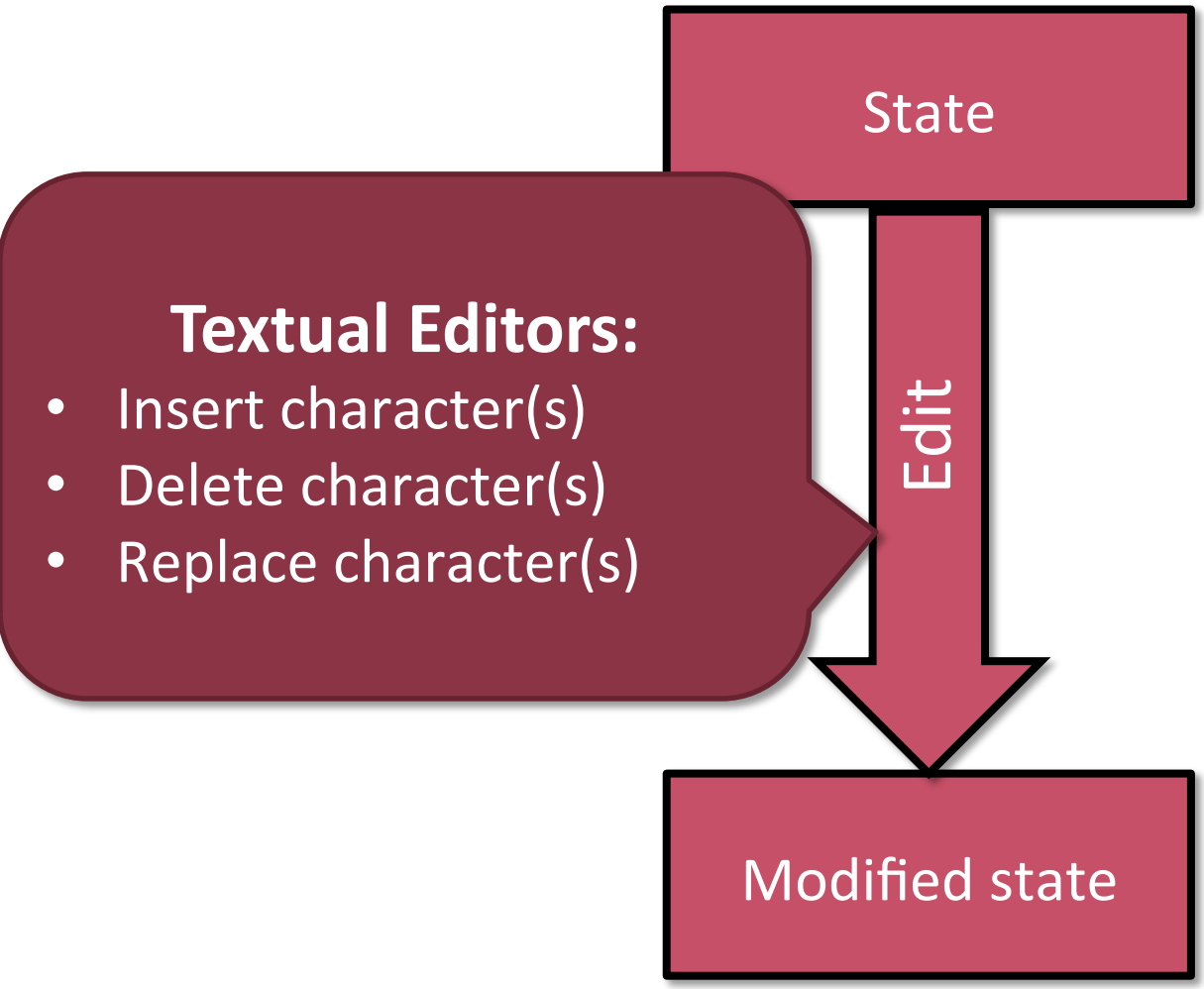
Comparison

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	

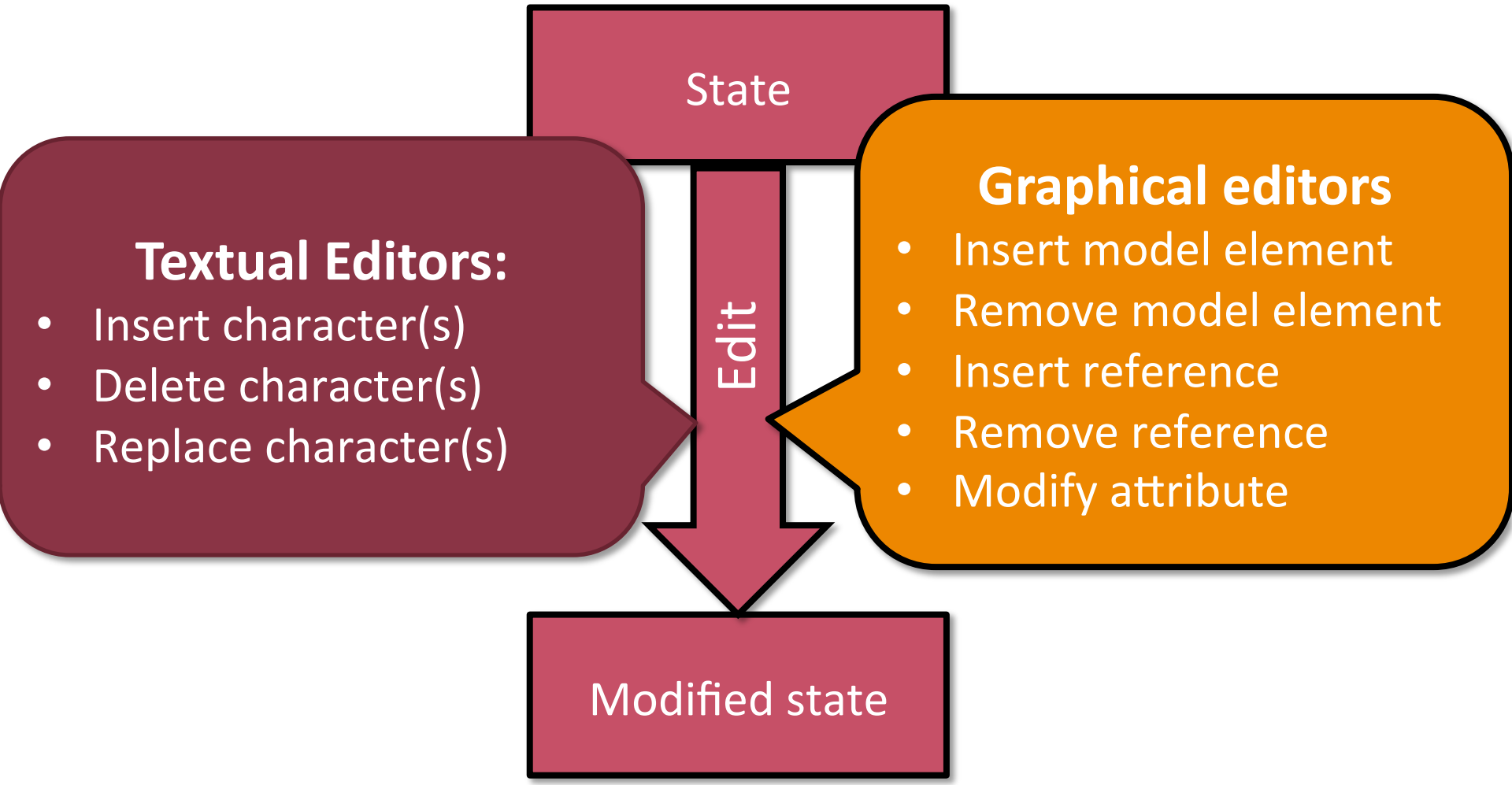
Editing



Editing


State
Edit
Modified state
Textual Editors:
• Insert character(s)
• Delete character(s)
• Replace character(s)

Editing


State
Edit
Modified state
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

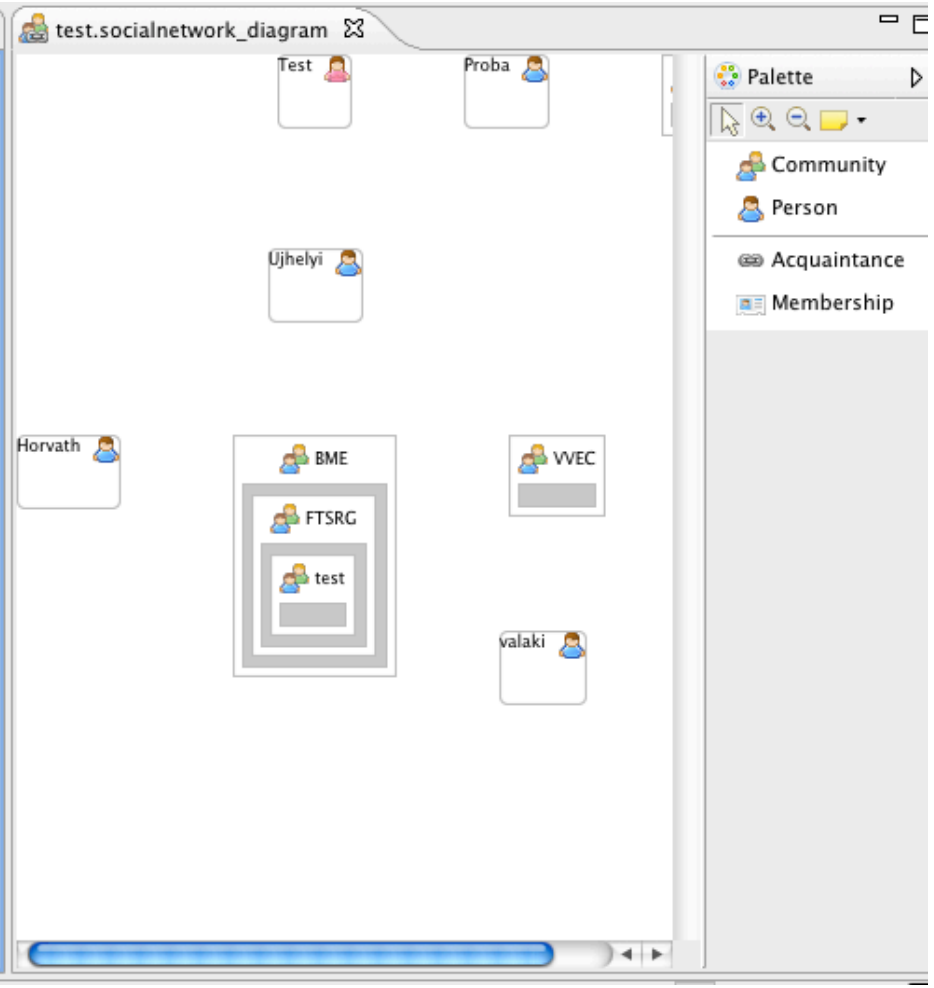
Question: textual or graphical?

- 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

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
}
```



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