Universität der Bundeswehr München

Syntax Specification by Graph Grammars and Meta-Models

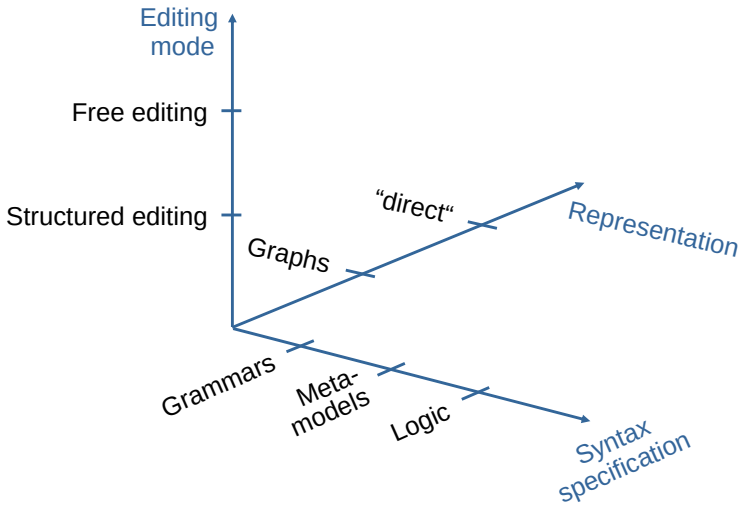
Mark Minas
Institute for Software Technology
Universität der Bundeswehr München
Germany

Mark.Minas@unibw.de

Outline

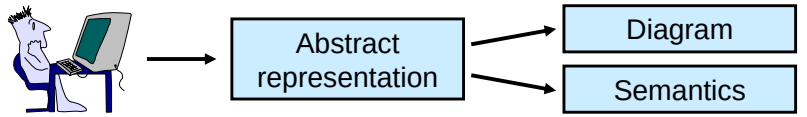
- (Some) Dimensions of Visual Languages & Editors
- ***DiaGen***
  - Editor architecture
  - Hypergraph **grammar-based** specification and diagram analysis
- ***DiaMeta***
  - Editor architecture
  - **Metamodel-based** specification and diagram analysis
- Conclusions

(Some) Dimensions of Visual Languages & Editors



Editing Modes of Diagram Editors

- **Structured editing**

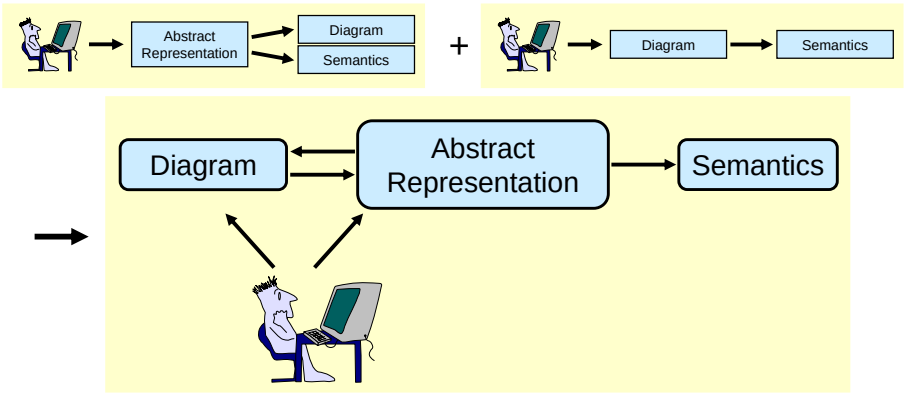


  - ☺ Easy to build
  - ☺ Offers a set of editing operations
  - ☹ Restricts user
- **Free editing**

  - ☹ Requires expensive diagram analysis
  - ☺ Offers maximum editing freedom
  - ☹ Doesn't offer guidance

Editing Modes of Diagram Editors

- Combination of **structured** and **free** editing
  - ☺ Offers a set of editing operations
  - ☺ Offers maximum editing freedom
  - ☹ Requires expensive diagram analysis

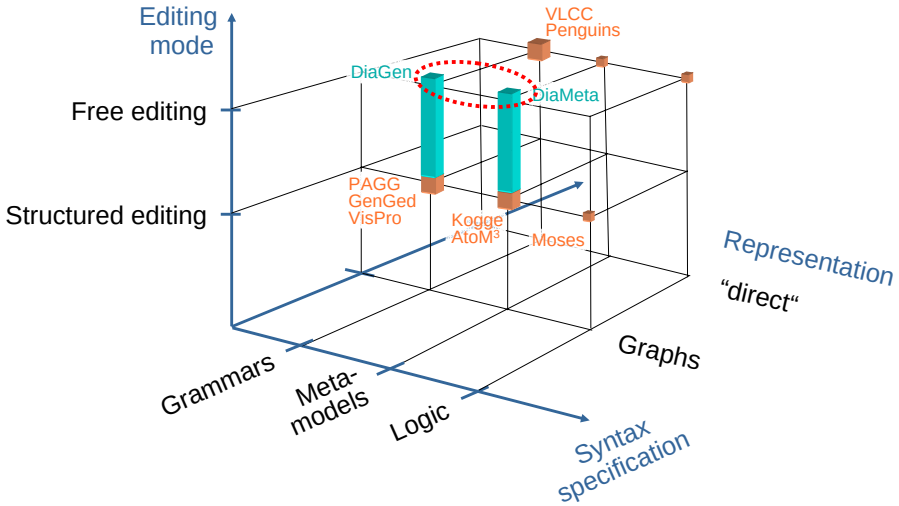


Syntax Specification Methods for Visual Languages

- **"Direct" syntax specification**
  - **VLCC** (Costagliola et al.)
    Positional Grammars
  - **Penguins** (Chok, Marriott)
    Constraint Multiset Grammars
  - **Many others…**
- **Graphs as abstract representation**
  - **PAGG** (Göttler)
    Graph transformations
  - **GenGed** (Bardohl)
    Graph transformations
  - **VisPro** (Zhang)
    Confluent graph grammars
  - **DiaGen/DiaMeta**
    Hypergraph grammars & Metamodels
  - **Atom³** (de Lara, Vangheluwe)
    Metamodels
  - **Kogge** (Ebert et al.)
    Graph schemas (=Metamodels)
  - **Moses** (Janneck, Esser)
    Assertions
  - **Many others…**

Syntax Specification Methods for Visual Languages

- (Some) Dimensions of visual languages & editors



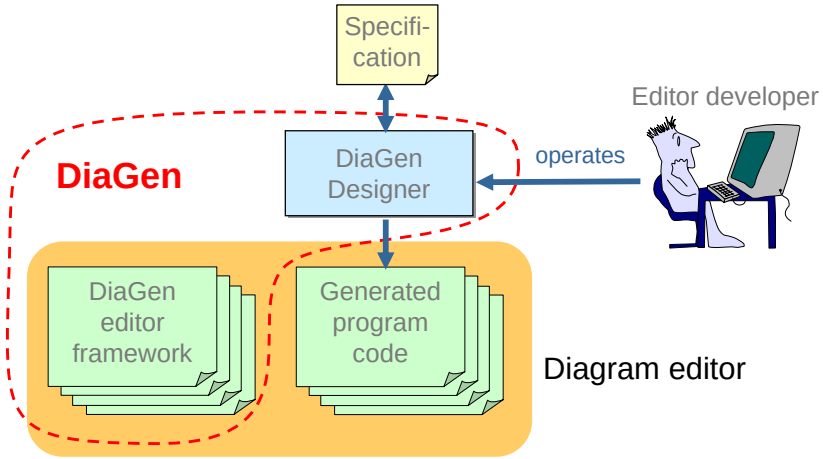
Outline

- (Some) Dimensions of Visual Languages & Editors
- ***DiaGen***
  - Editor architecture
  - Hypergraph **grammar-based** specification and diagram analysis
- ***DiaMeta***
  - Editor architecture
  - **Metamodel-based** specification and diagram analysis
- Conclusions

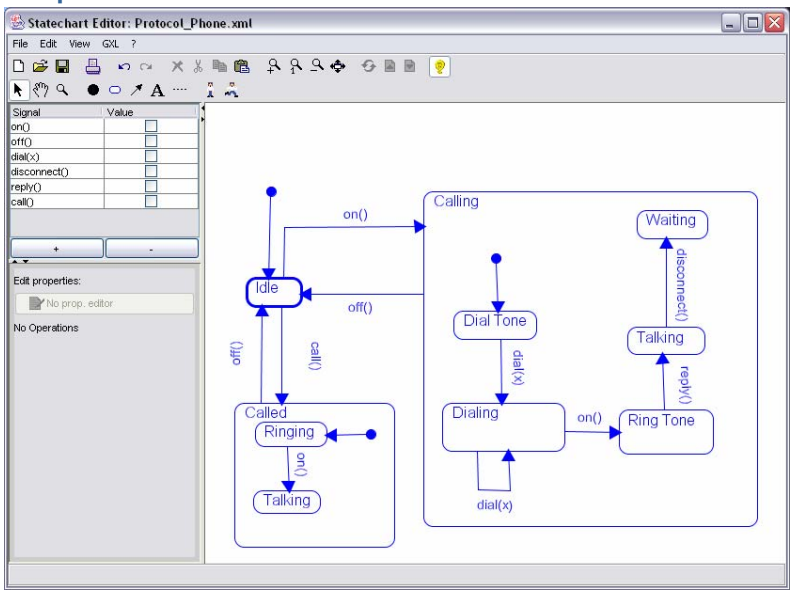
DiaGen

- *Rapid Prototyping* tool for creating diagram editors from a specification
- Free-hand editing:
  - Drawing tool
  - Lexical, syntactic, and semantic analysis
- Structured editing (optional)
- Automatic layout (optional)
  even for free-hand editing
- Support of hierarchical diagrams (optional)
- Unparsing of diagrams,
  i.e. external format → diagram (optional)
- Semantic zooming (optional)
- Java

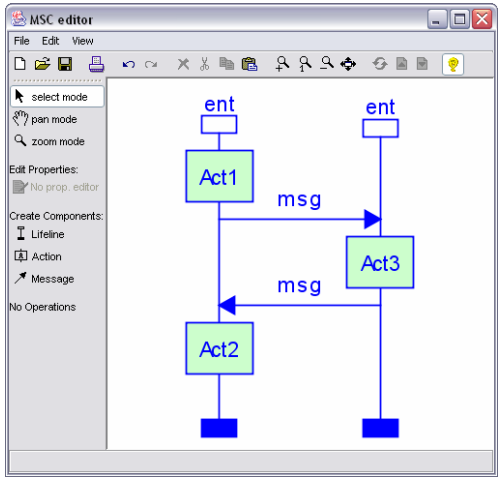
Generating diagram editors with DiaGen



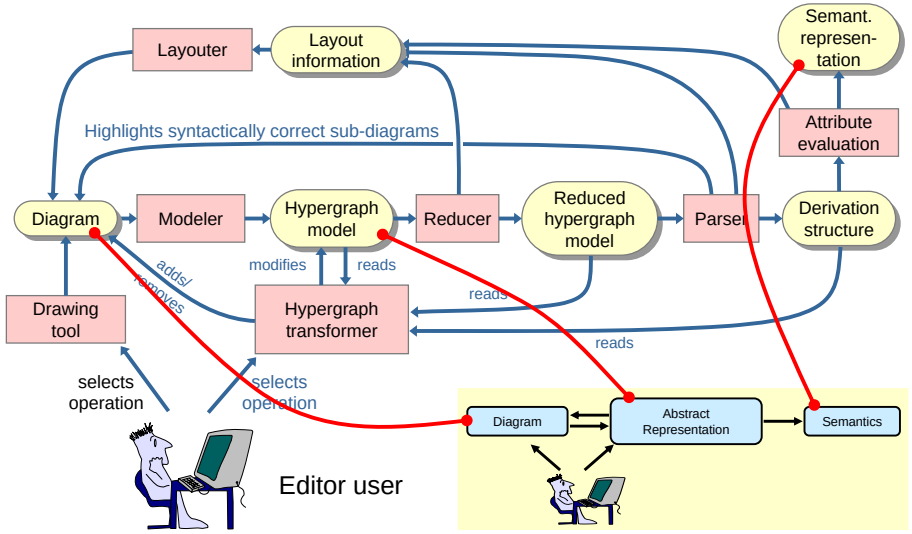
Example: Statecharts



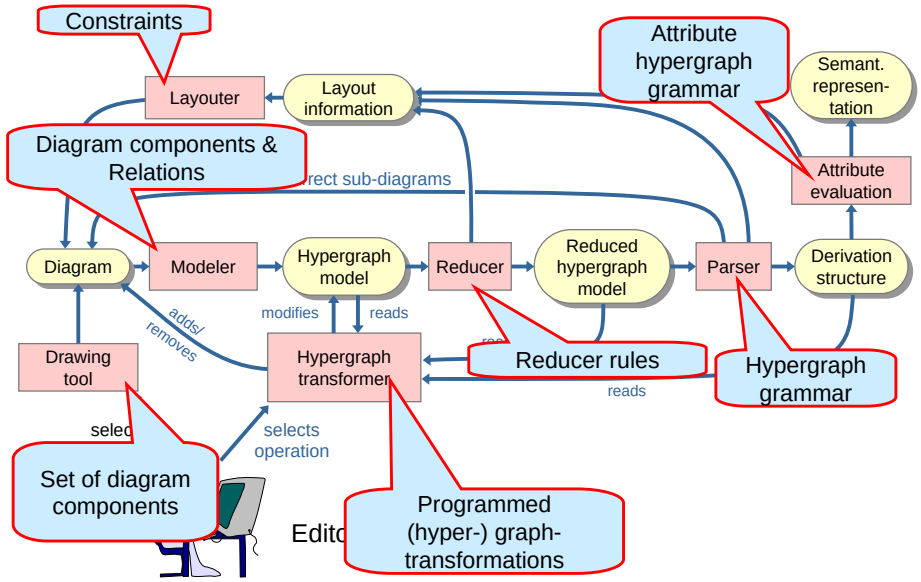
Example: Message Sequence Charts (MSC)



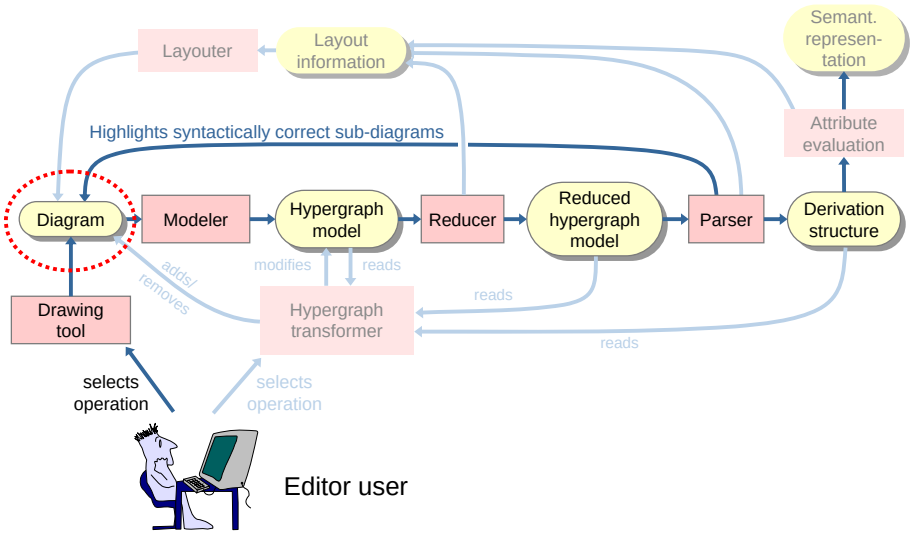
DiaGen: Editor Architecture



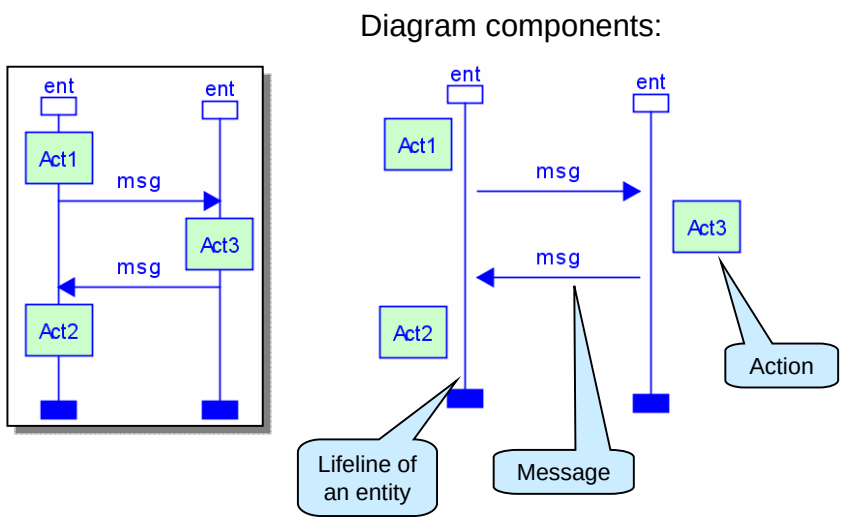
DiaGen: Editor Architecture



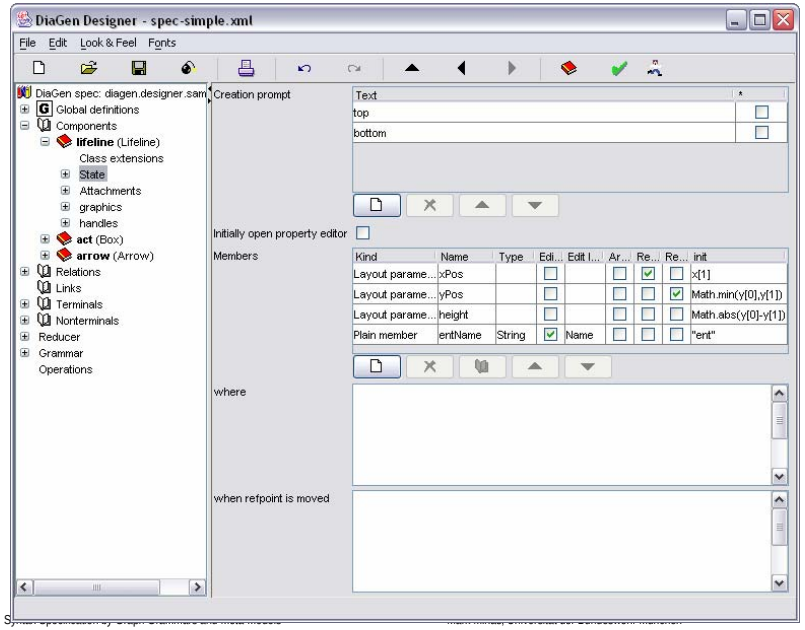
DiaGen: Editor Architecture



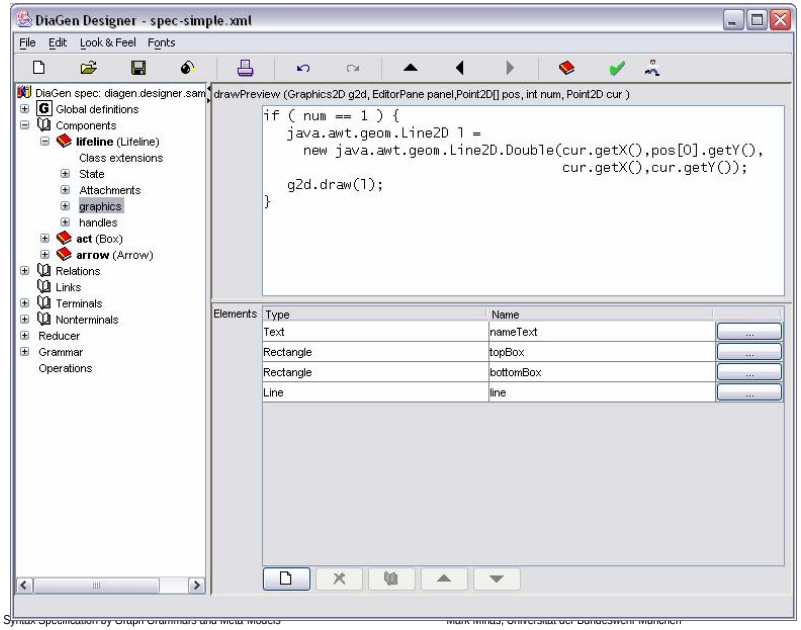
Example: Message Sequence Charts (MSC)



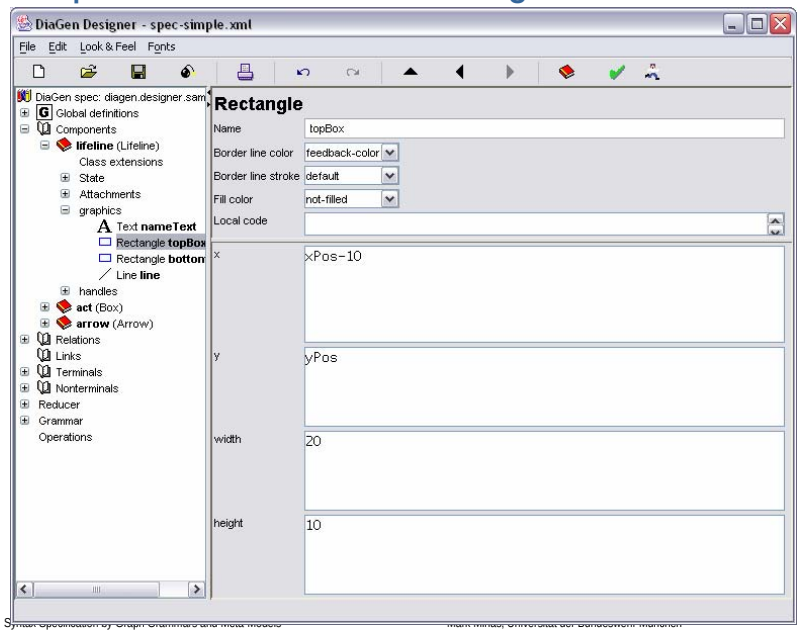
...specified in the DiaGen Designer



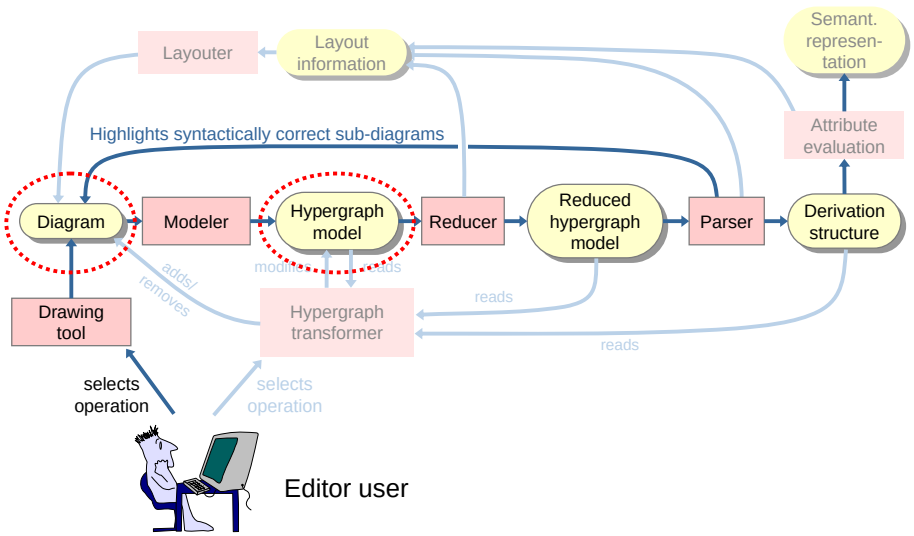
...specified in the DiaGen Designer



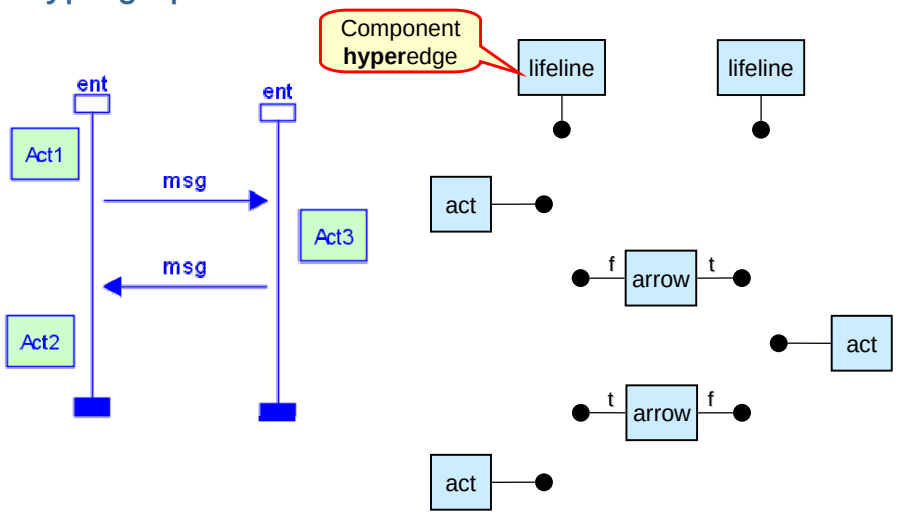
...specified in the DiaGen Designer



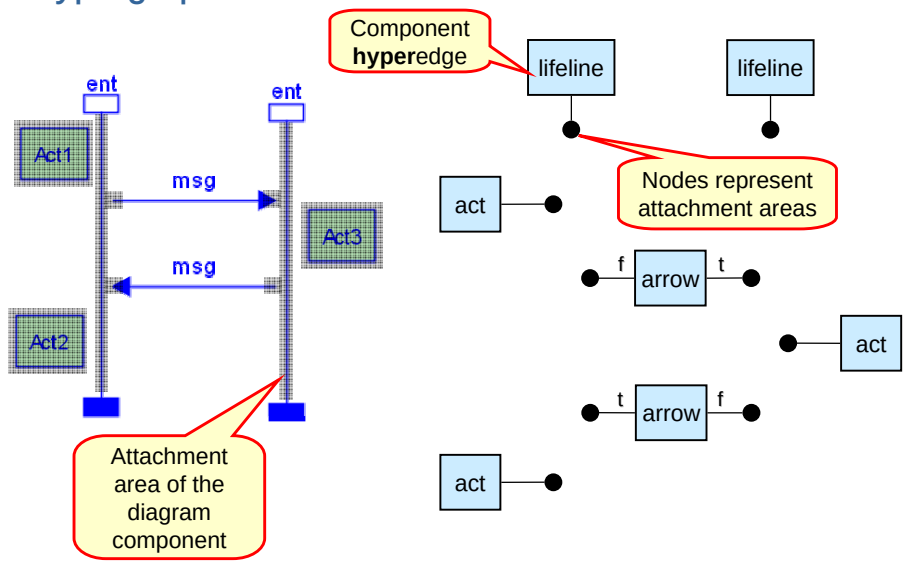
DiaGen: Editor Architecture



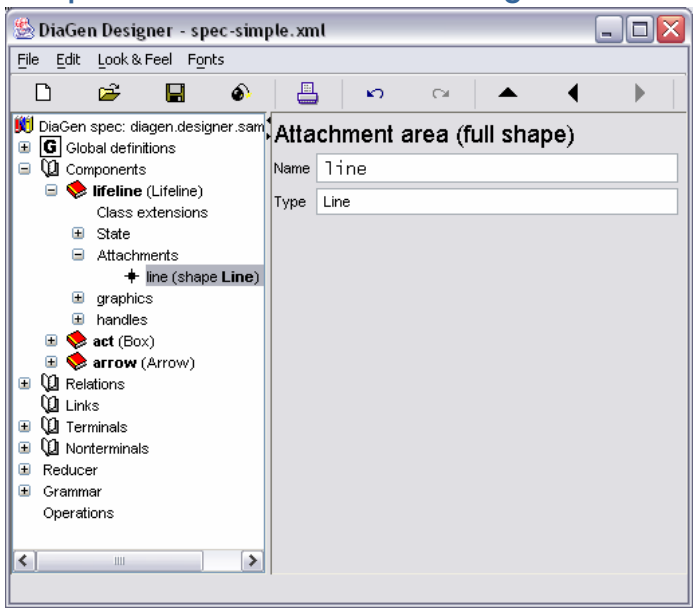
Hypergraph Model



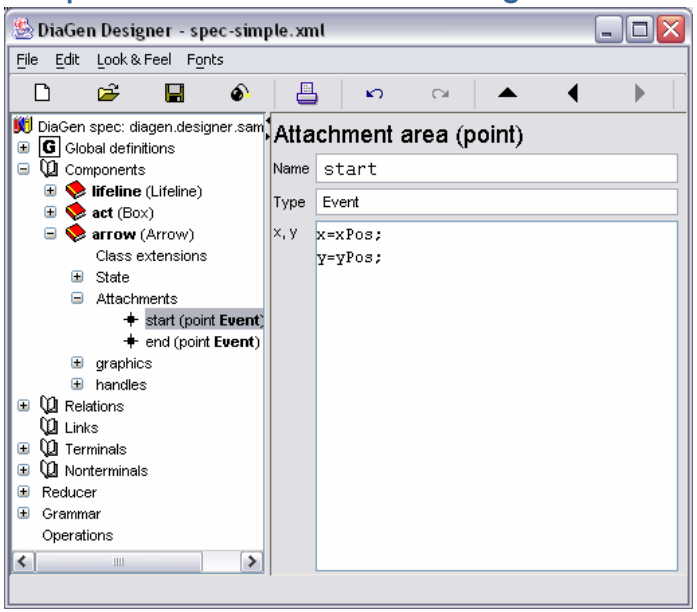
Hypergraph Model



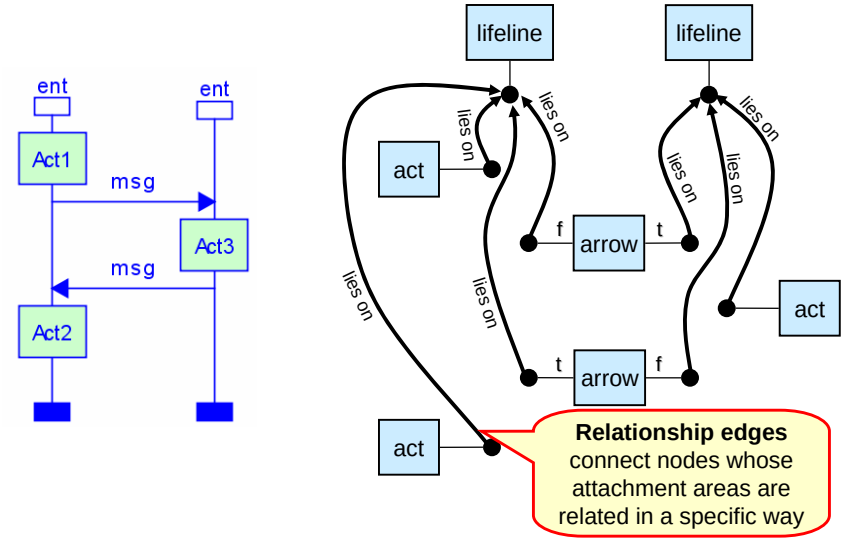
...specified in the DiaGen Designer



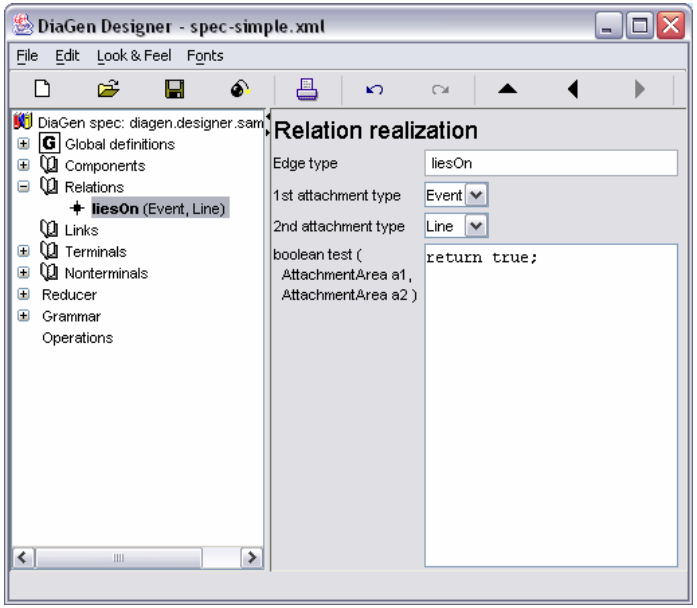
...specified in the DiaGen Designer



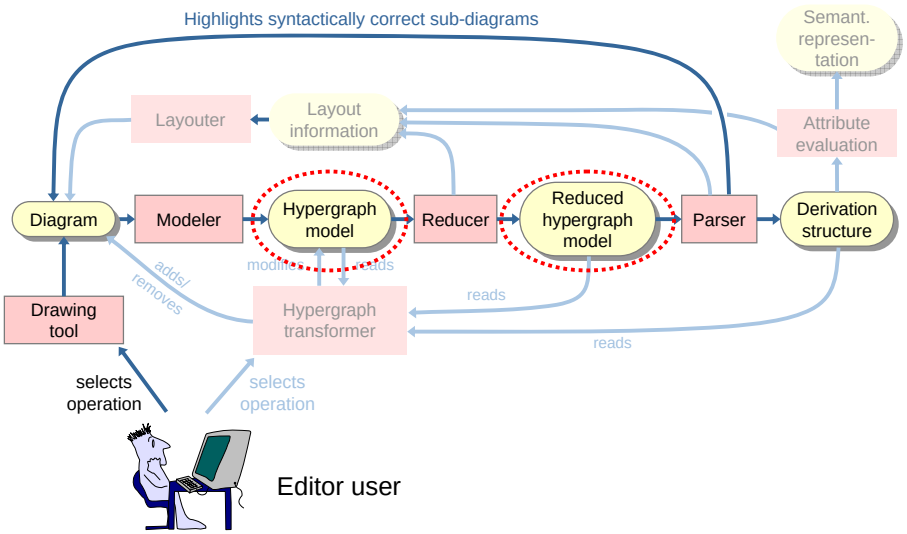
Hypergraph Model with Relationship Edges



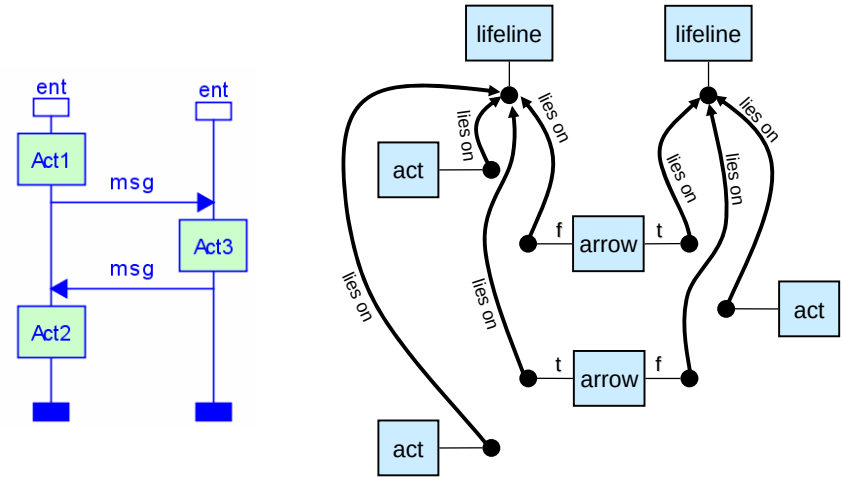
...specified in the DiaGen Designer



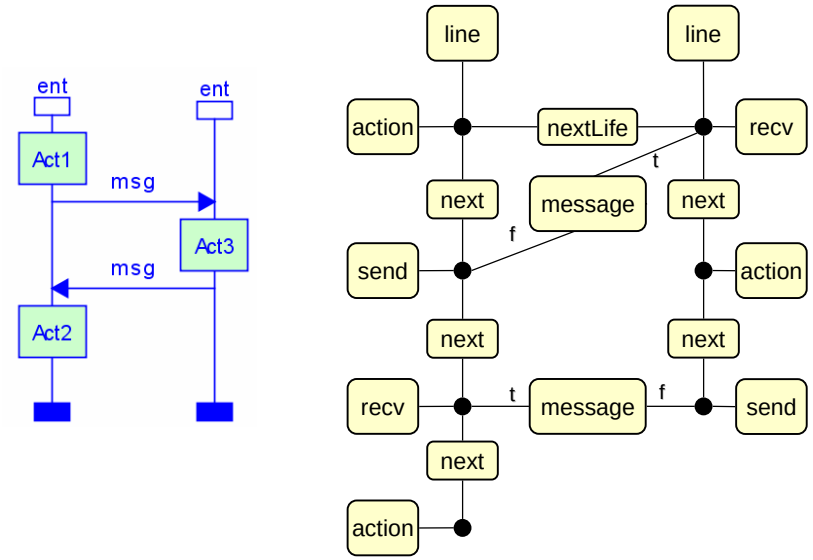
DiaGen: Editor Architecture



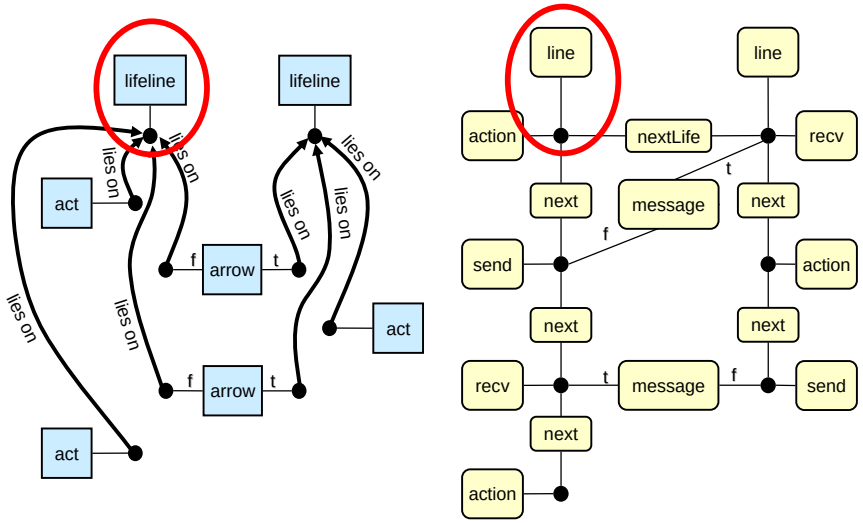
Hypergraph Model with Relationship Edges



Reduced Hypergraph Model

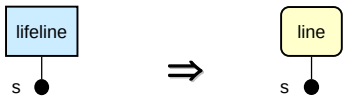


"Reducing" the Hypergraph Model



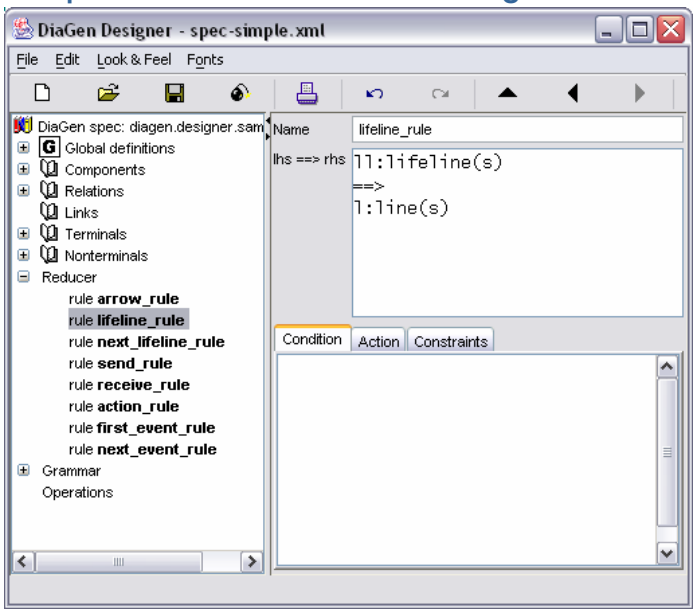
"Reducing" the Hypergraph Model

- Reducer rule

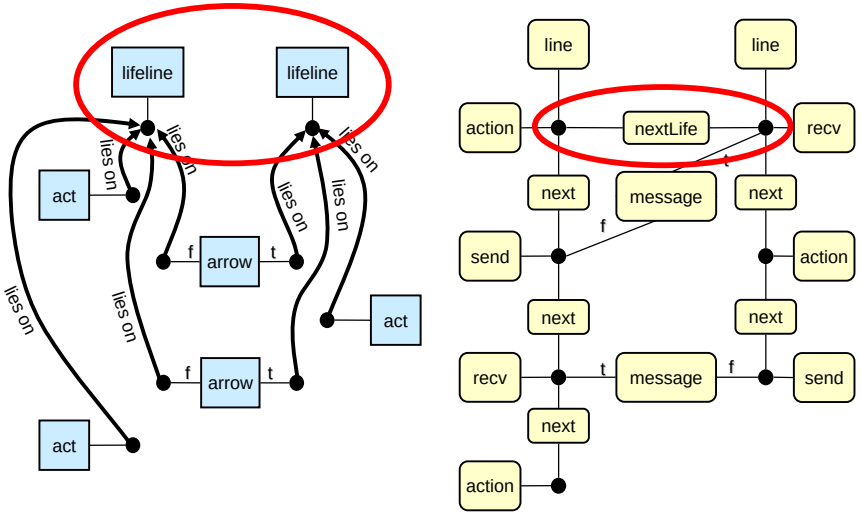


  - For each *lifeline* edge and its visited node, create a *line* edge together with its visited node in the RHGM
  - Actually (for each reducer rule!!):
    - Don't create a new node in the RHGM if the HGM node has a corresponding RHGM node already. Use this RHGM node instead!
  - Textual representation:
    `lifeline(s) ==> line(s)`

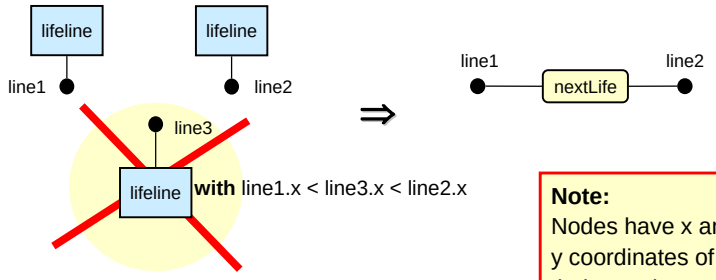
...specified in the DiaGen Designer



“Reducing“ the Hypergraph Model



“Reducing“ the Hypergraph Model



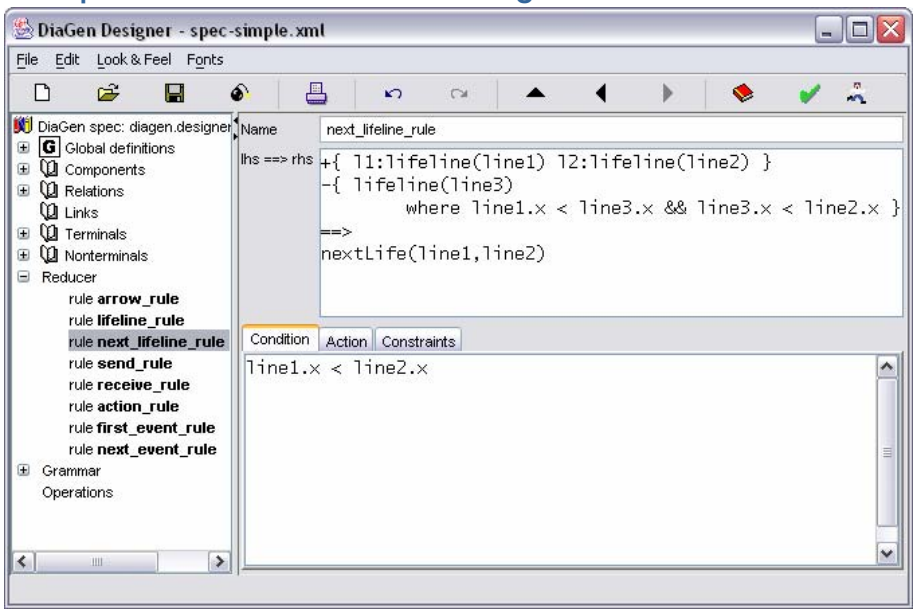
Condition: line1.x < line2.x

Note:
Nodes have x and y coordinates of their attachment areas

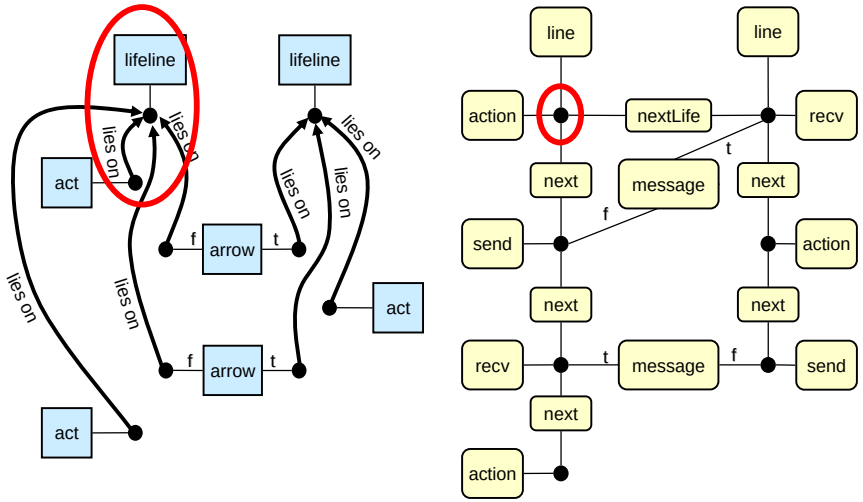
Textual representation:

```
lifeline(line1) lifeline(line2)
-{ lifeline(line3)
          where line1.x < line3.x && line3.x < line2.x }
==>
nextLife(line1,line2)
```

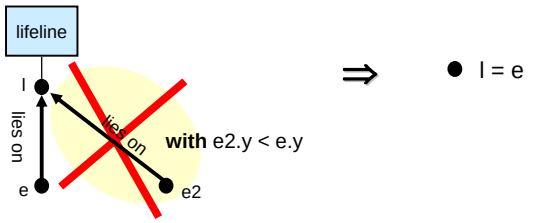

...specified in the DiaGen Designer



“Reducing“ the Hypergraph Model



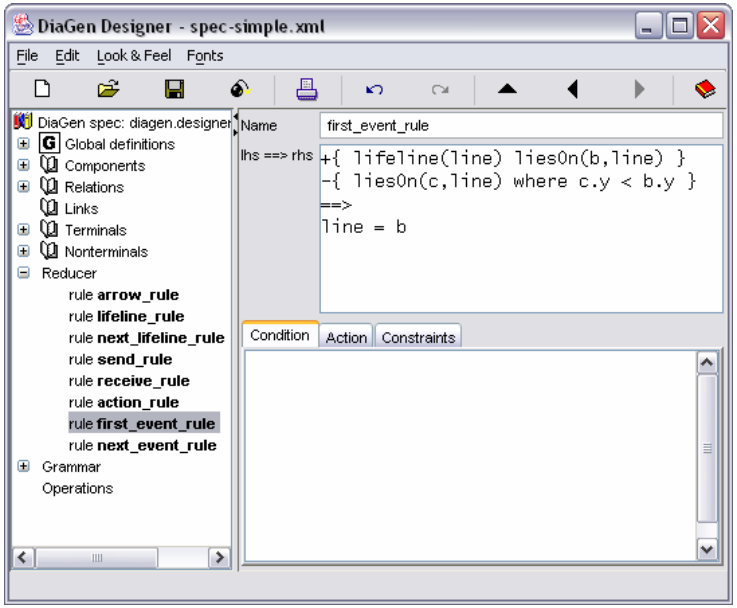
“Reducing“ the Hypergraph Model



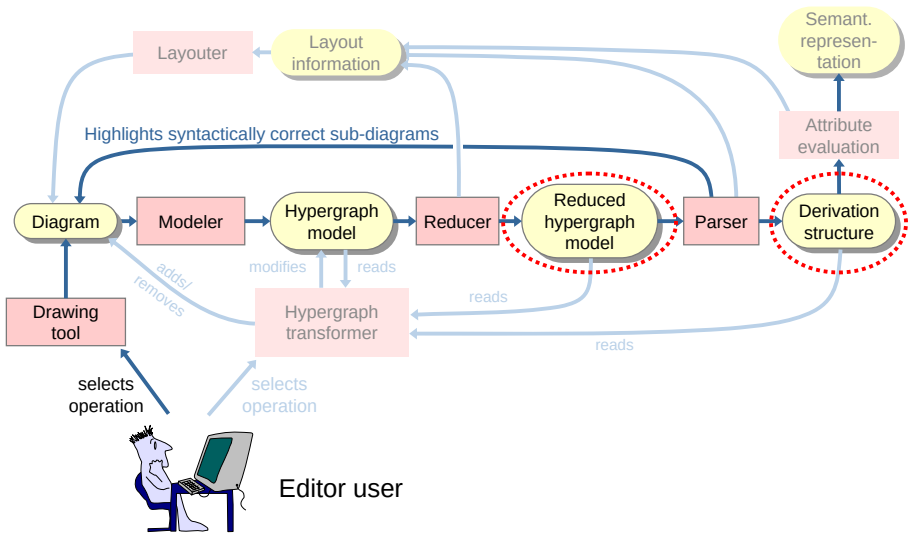
Textual representation:

```
lifeline(l) liesOn(e,l)
-{ liesOn(e2,l) where e2.y < e.y }
==>
e = l
```

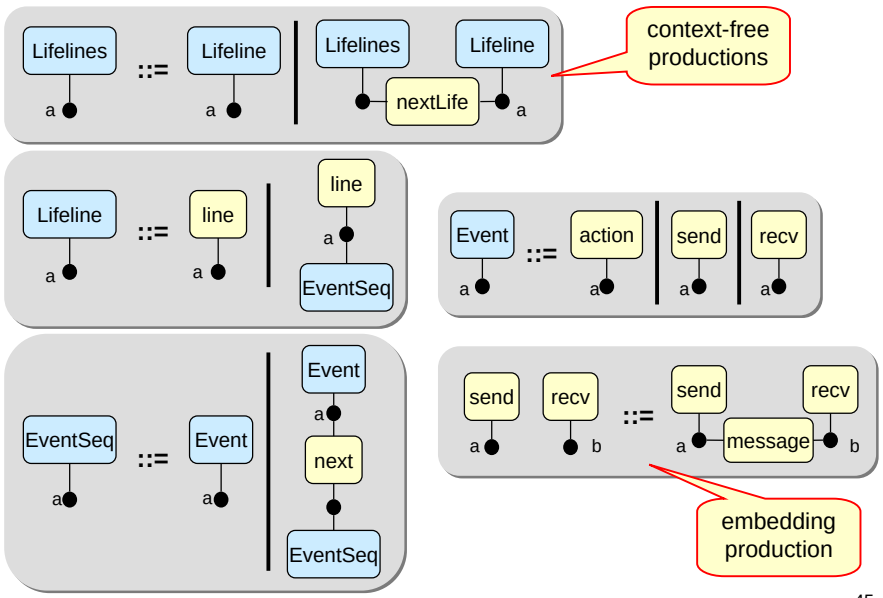

...specified in the DiaGen Designer



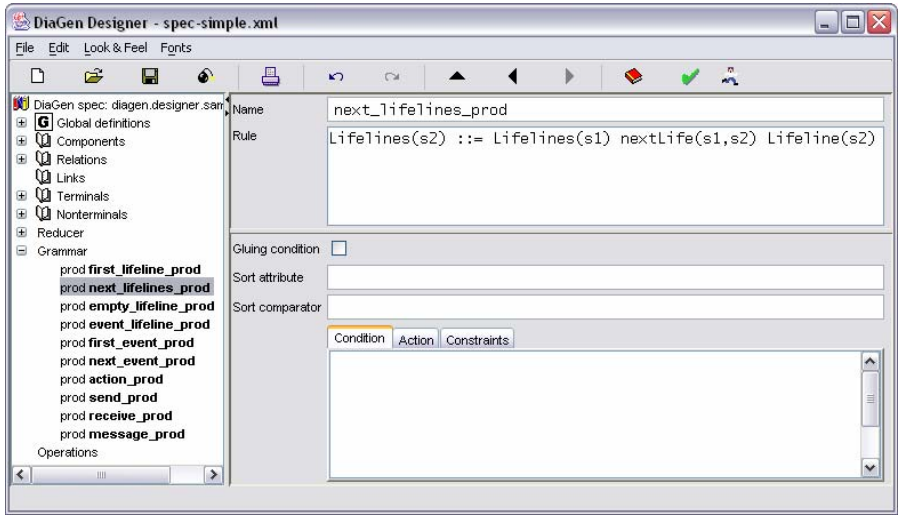
DiaGen: Editor Architecture



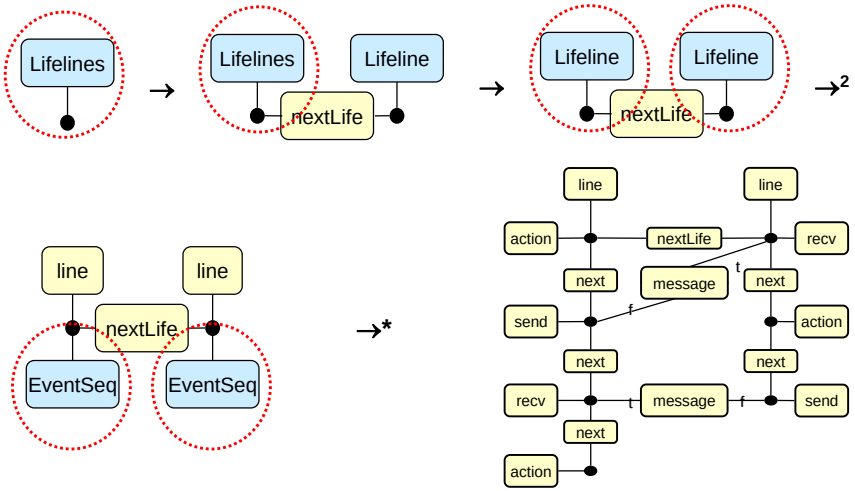
Hypergraph Grammar



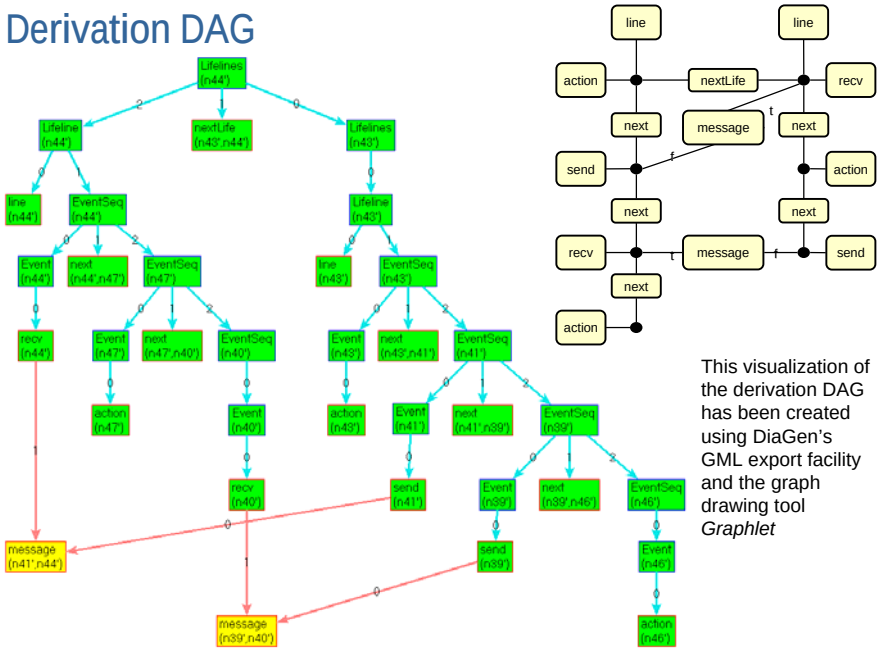
...specified in the DiaGen Designer



Hypergraph Grammar – Derivation Sequence



Derivation DAG



This visualization of the derivation DAG has been created using DiaGen's GML export facility and the graph drawing tool *Graphlet*

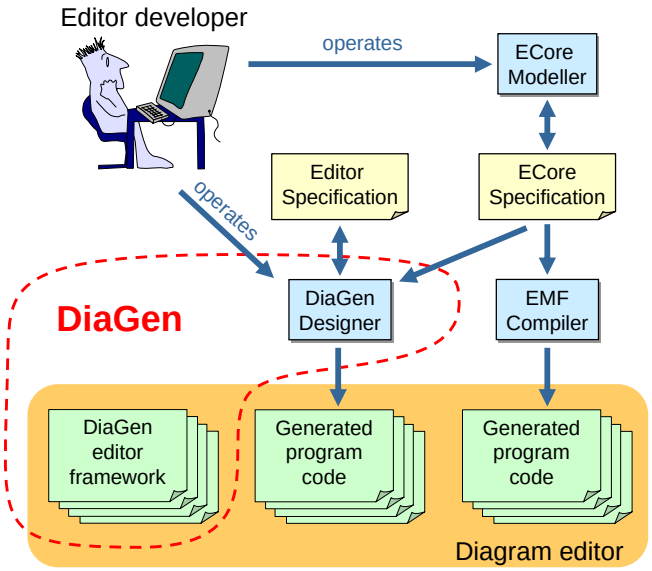
Outline

- (Some) Dimensions of Visual Languages & Editors
- ***DiaGen***
  - Editor architecture
  - Hypergraph **grammar-based** specification and diagram analysis
- ***DiaMeta***
  - Editor architecture
  - **Metamodel-based** specification and diagram analysis
- Conclusions

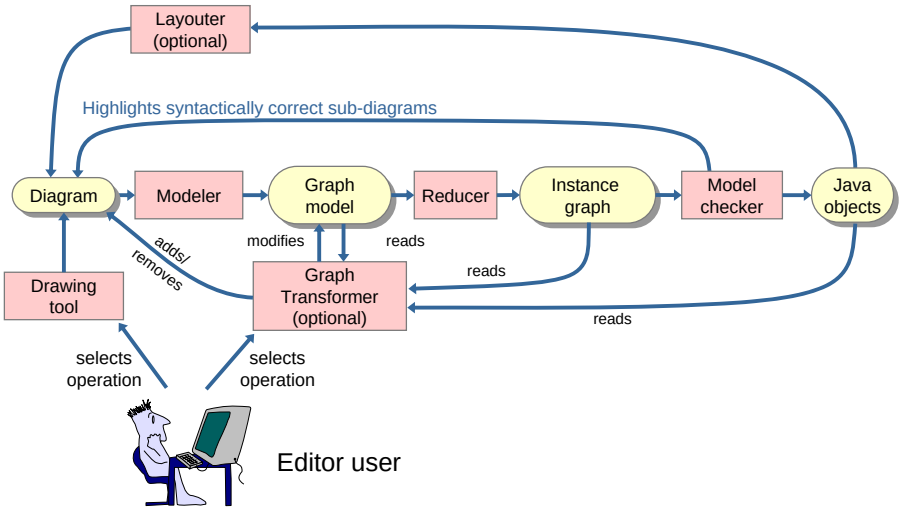
DiaGen with Metamodels

- **Observation:**
  - Most visual languages are graph-like
  - Most people are not used to used to syntax specification using (graph) grammars
  - However: Most people are used to class diagrams
- → Use metamodels as an alternative to (graph) grammars for language specification
- Implemented using EMF (Eclipse Modeling Framework)
- **Benefits**:
  - Fast language specification
  - Metamodel also specifies internal representations of edited diagrams

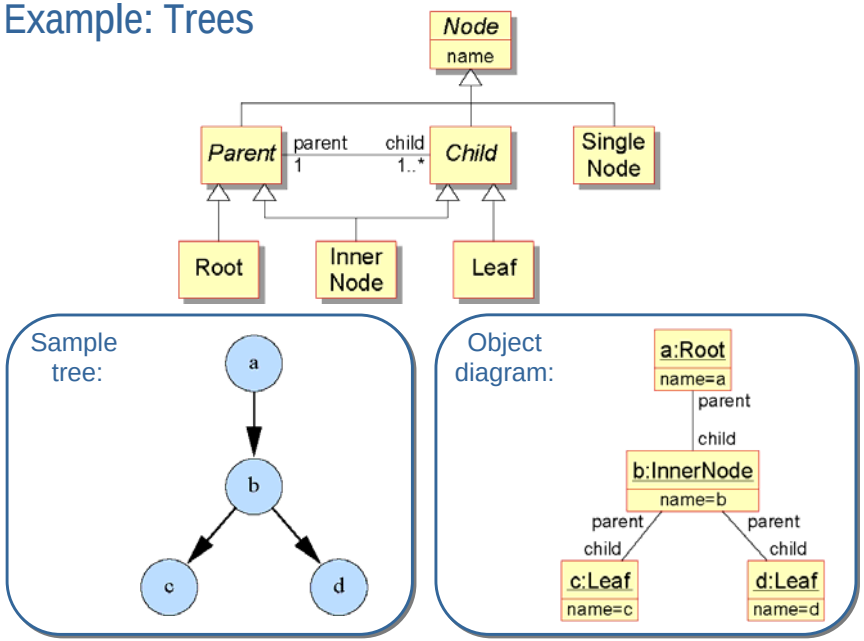
Generating diagram editors with DiaMeta (EMF)



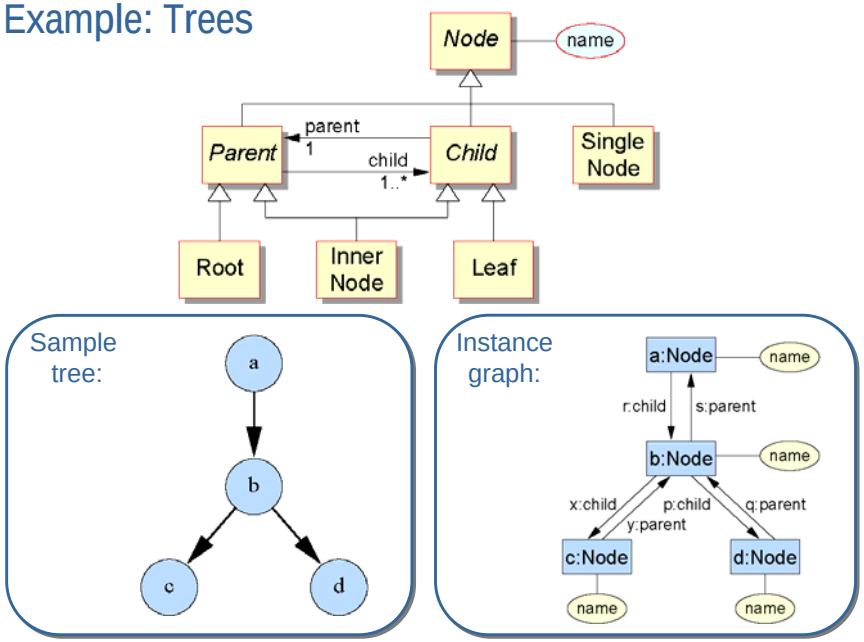
DiaMeta: Editor Architecture



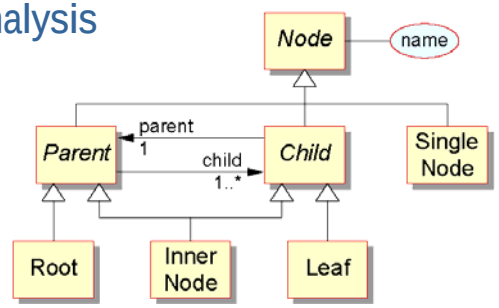
Example: Trees



Example: Trees

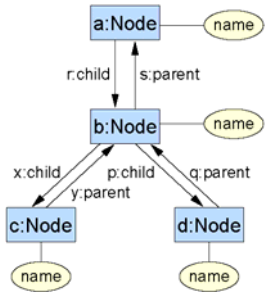


Syntax Analysis



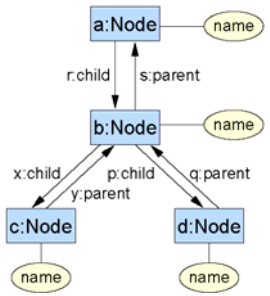
Graph homomorphism that obeys subclassing

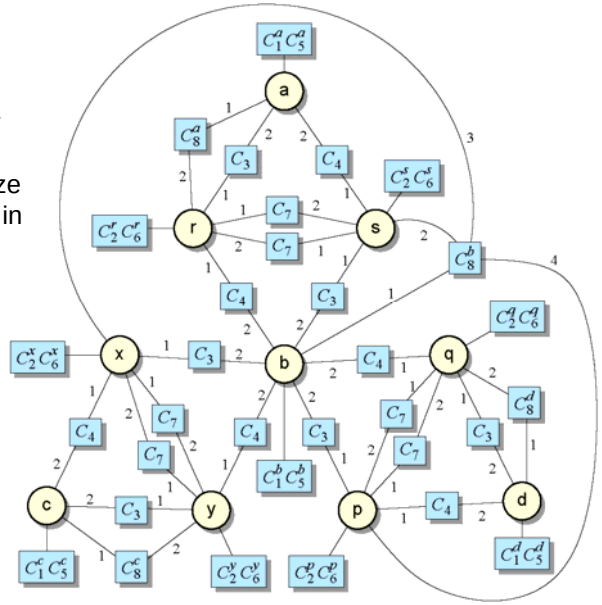
- Nodes are mapped to nodes
- Edges are mapped to edges
- Structure preservation
- Cardinality constraints not violated



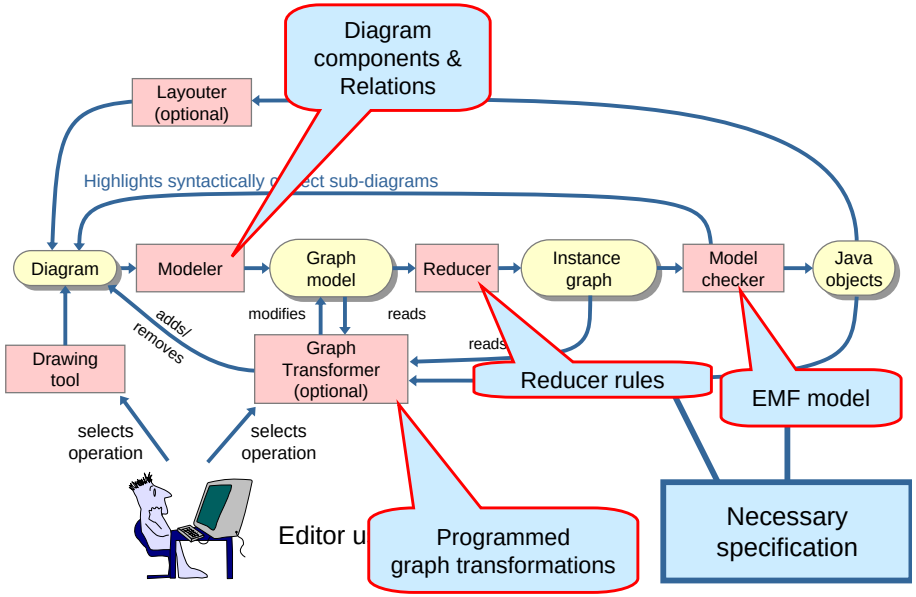
Syntax Analysis

- Search for Morphism: **C**onstraint **S**atisfaction **P**roblem
- Node / Arc consistency
- Linear in #constraints
- Quadratic in domain size
- Backtracking needless in practical cases



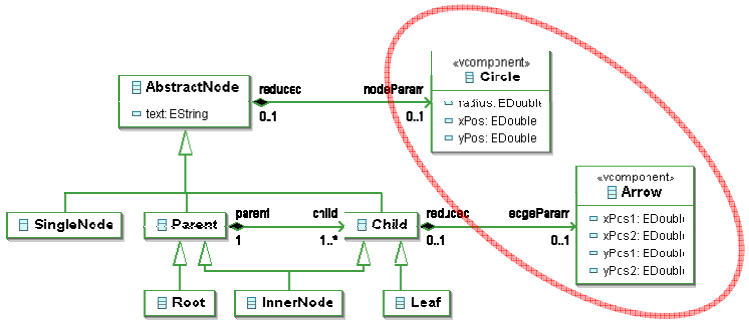


DiaMeta: Editor Architecture



Example: Trees

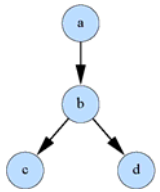
- EMF model (drawn with Omondo EclipseUML)

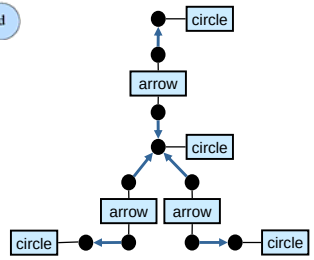


Represent diagram components together with their states

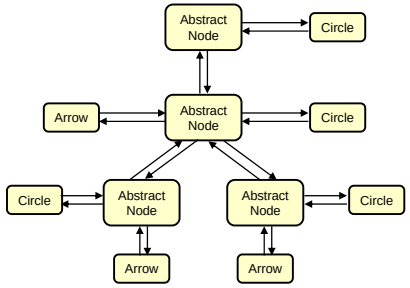
Example: Trees

- Reducer





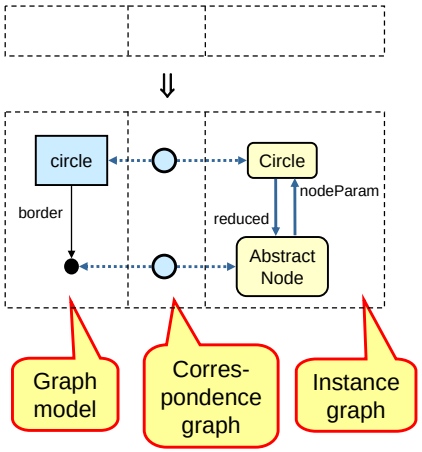
Graph model

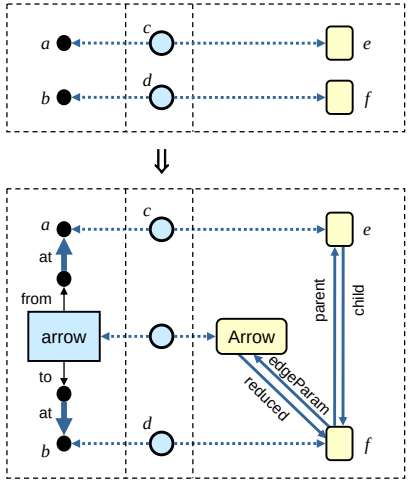


Instance graph

Example: Trees

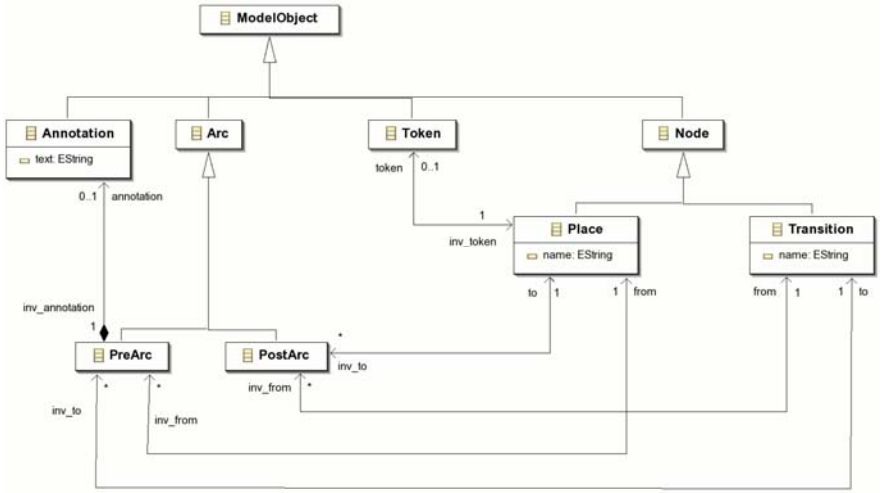
- Reducer rules
  = triple graph grammar rules





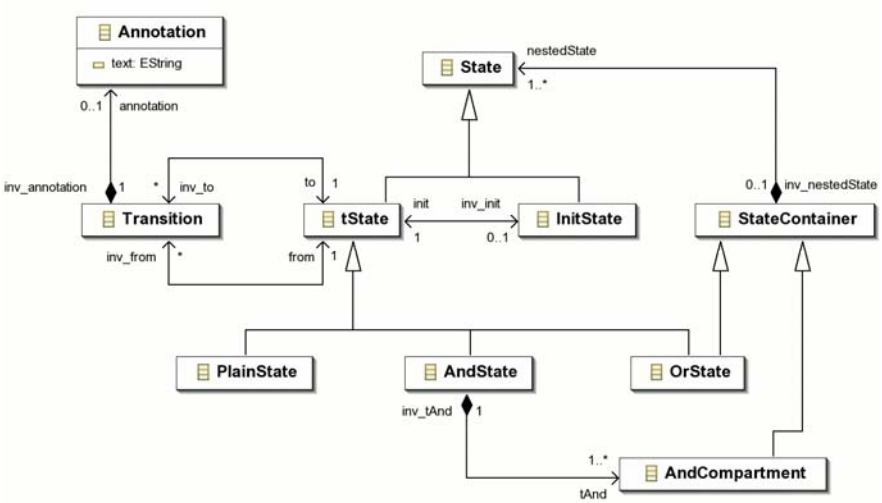
Example: Petri-nets

- EMF model (drawn with Omondo EclipseUML)



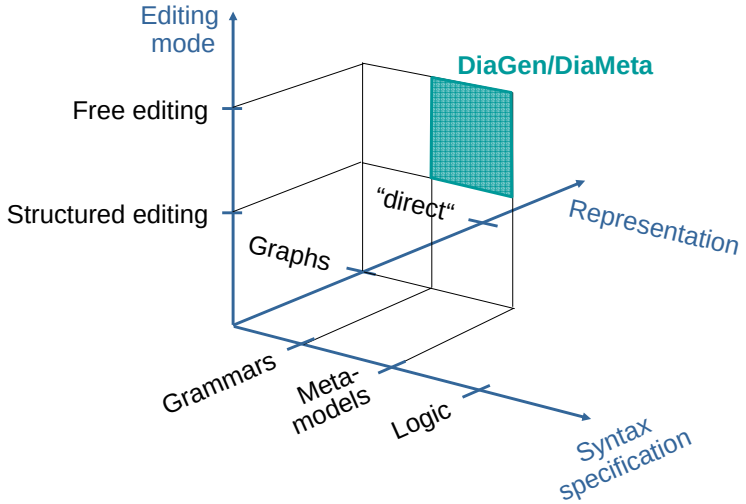
Example: Statecharts

- EMF model (drawn with Omondo EclipseUML)



Conclusions

- (Some) Dimensions of visual languages & editors



Future Work

- Using MOF as an alternative to EMF
- Diagram layout
- Simplifying free editing with automatic layout
- Sketching
- Model transformation
- Improving structured editing

Improving structured editing

- Operate on instance graph instead of graph model