Domain-Specific Languages (DSLs)

Juan de Lara, Elena Gómez, Esther Guerra
{Juan.deLara, MariaElena.Gomez, Esther.Guerra}@uam.es
Computer Science Department
Universidad Autónoma de Madrid

Masters: I2TIC and Formal methods

Index

- **Introduction.**

  - **Syntax.**
  - **Semantics.**
  - **Examples.**
- Types of modelling environments.
- Technologies to build DSLs.
- Bibliography.

Domain-Specific Languages (DSLs)

- Languages oriented to a particular application domain or problem (in contrast to general-purpose languages).
- They capture the knowledge and experience in a specific application area.
- High-level, expressive, powerful primitives.
- Premise: DSLs enhance productivity compared to using general-purpose languages.
- DSLs are extensively created/used in MDE solutions.

Problem domain vs Solution domain

Domain-Specific Languages

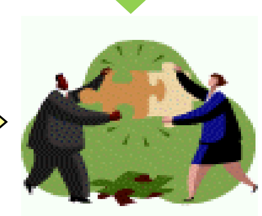
- oriented to users way of thinking
- smaller semantic gap to problem

problem space

(semi)automatic transformation

General-Purpose Languages

- oriented to developers way of thinking
- need to transform into technical domain



solution space

Types of Domain-Specific Languages

- **Internal or embedded**: they use the infrastructure of an existing host language (e.g., Ruby, UML profiles).
  - Shorter development time
  - Same concrete syntax as the host language

- **External**: they are built from scratch.
  - Flexibility on the concrete syntax of the language
  - Costly implementation (requires implementing parser, syntactic analyzer, interpreter or compiler, editing environment, etc.)
  - …but there are frameworks that facilitate their development, like Sirius (for graphical DSLs) or Xtext (for textual DSLs)

External Domain-Specific Languages

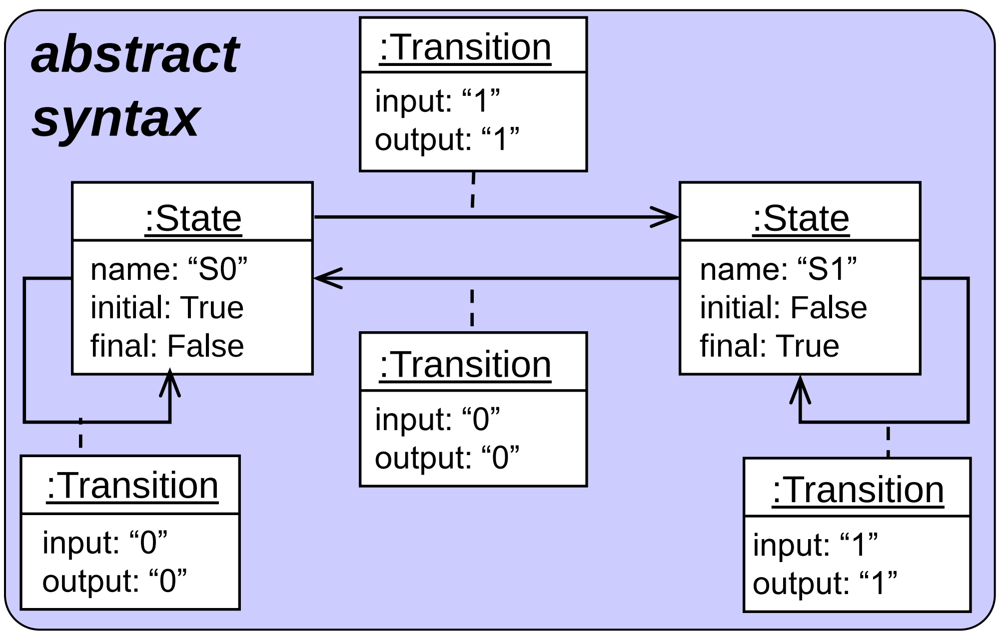
- DSLs can be graphical, textual, or a combination of text and graphics such as:
  - OCL + UML
  - Action language of UML
  - Languages including mathematical expressions

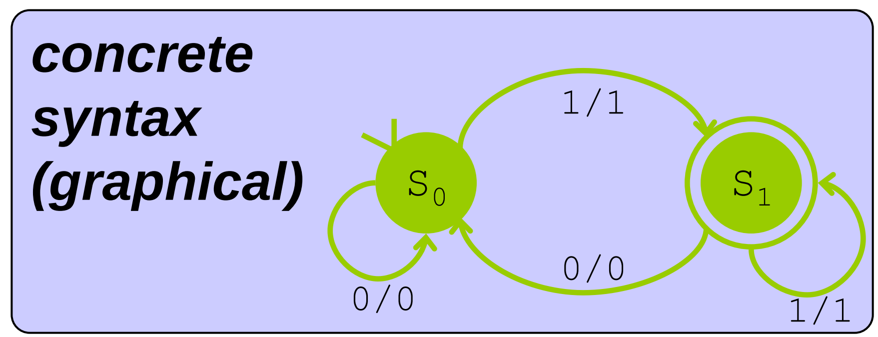
- Multi-view language: set of diagrams describing different aspects of a system.

- Combined with code generators and simulators.

Syntax

- **Abstract syntax**: language concepts, relations and attributes. It can be defined using a meta-model or a creation graph grammar.
- **Concrete syntax**: visualization of the abstract syntax elements.
  - Not necessarily a 1-to-1 mapping
  - Spatial relationships (e.g. containment, adjacency)
  - Spatial constraint languages (e.g. QOCA, https://www.swmath.org/software/756)





concrete syntax (textual)

States:
S0 {**initial**},
S1 {**final**}

Transitions:
S0 = 0/0 => S0
S0 = 1/1 => S1
S1 = 0/0 => S0
S1 = 1/1 => S1

Concrete syntax

Meta-modelling

- The concrete syntax can be given as graphical attributes of the classes and associations.

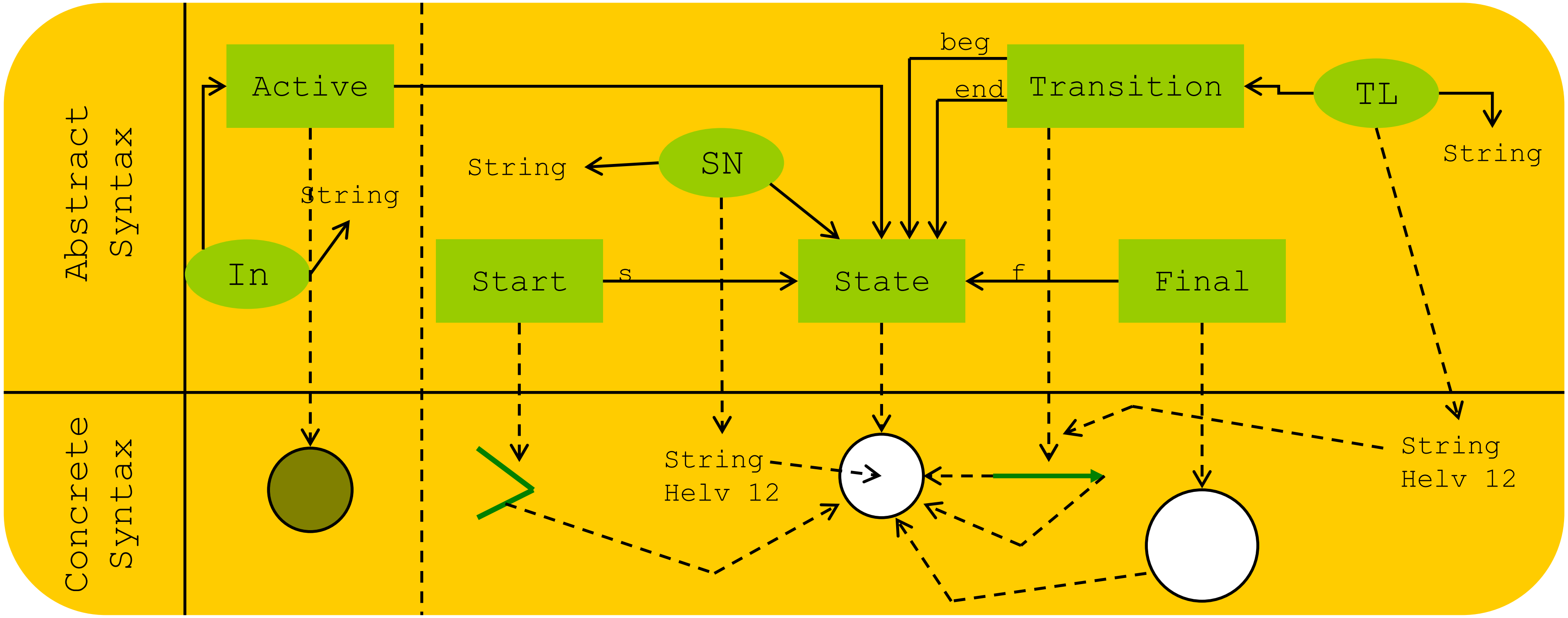
- For relations *n-to-m*, this can be very restrictive:
  - A meta-model for the concrete syntax and a meta-model for the abstract syntax. Transformations between them.

- Spatial relations, e.g. “contained”, “aligned with”, “touches”, etc.

Concrete syntax

Creation grammars

- Rules can use symbols of the alphabet of the concrete syntax.
- GenGED: it uses editor of symbols + constraint satisfaction system.



Semantics

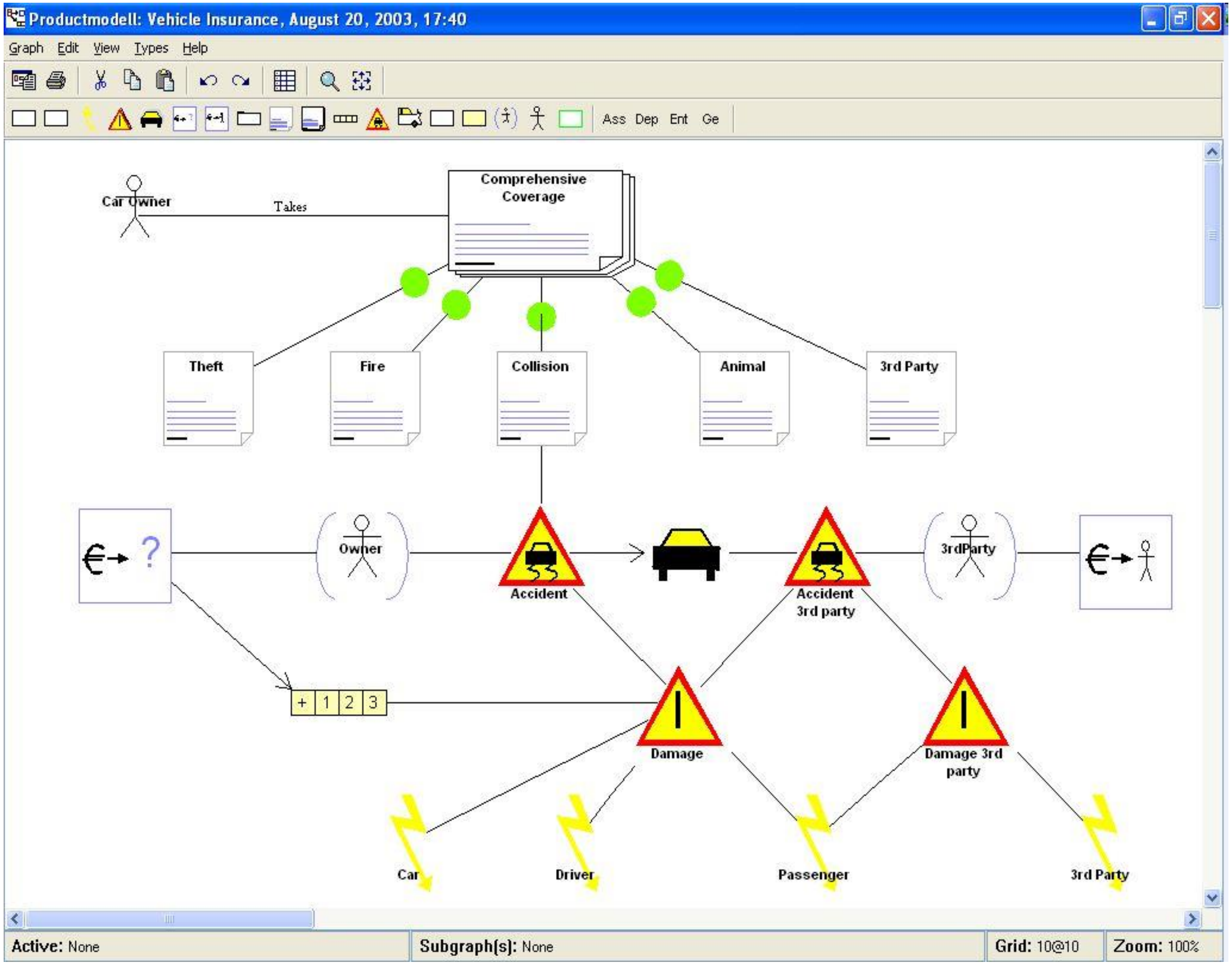
- **Static semantics**: Additional constraints.
  - Usually described using a constraint language like OCL
  - Is it semantics or syntax?

- **Operational semantics**: How to execute the model (simulator or “virtual machine” for the language).
  - Graph transformation, in-place model transformation techniques
  - A programming language

- **Denotational semantics**: Meaning of each construction in terms of a different formalism.
  - Model-to-model transformation
  - Code generation

Examples

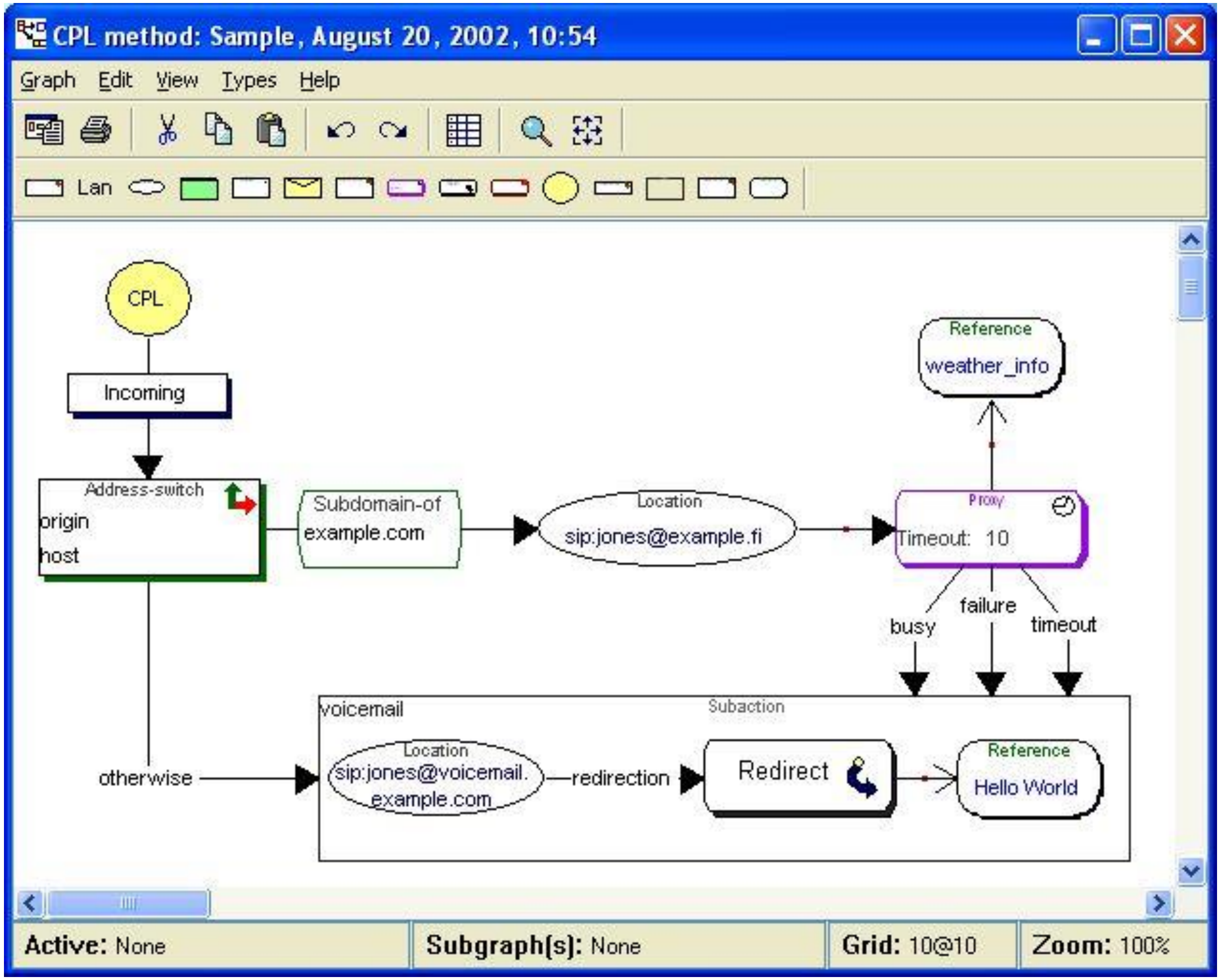
- Expert domain concepts.
- Simple code generation.
- Valid in well-known domains.
- Usable by non-programmers.



Ensurance company / J2EE

Examples

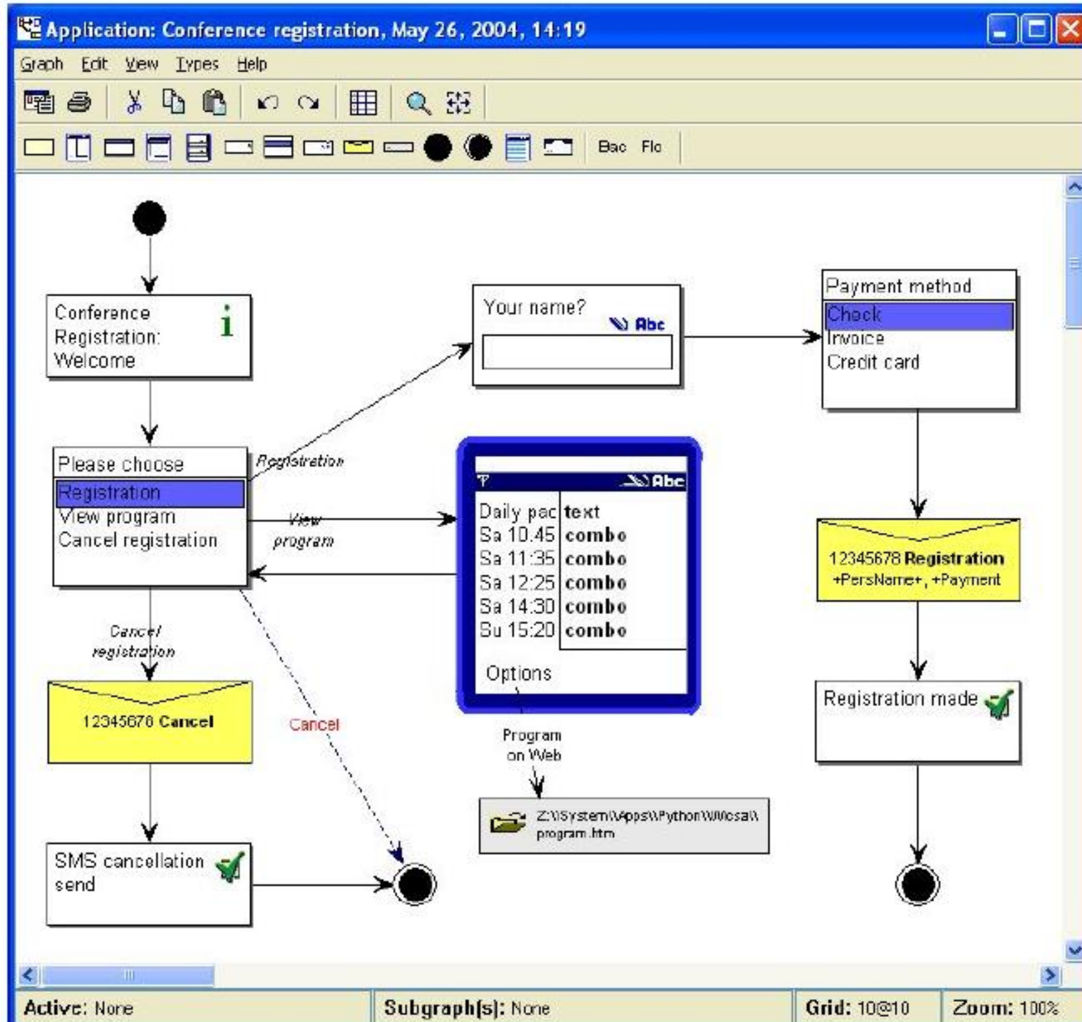
- Programming concepts.
- Static part is easy (data structures).
- In the limit, visual notation for programming language.
- Danger of low level of abstraction, small increase in productivity.



Internet Telephony / CPL

Examples

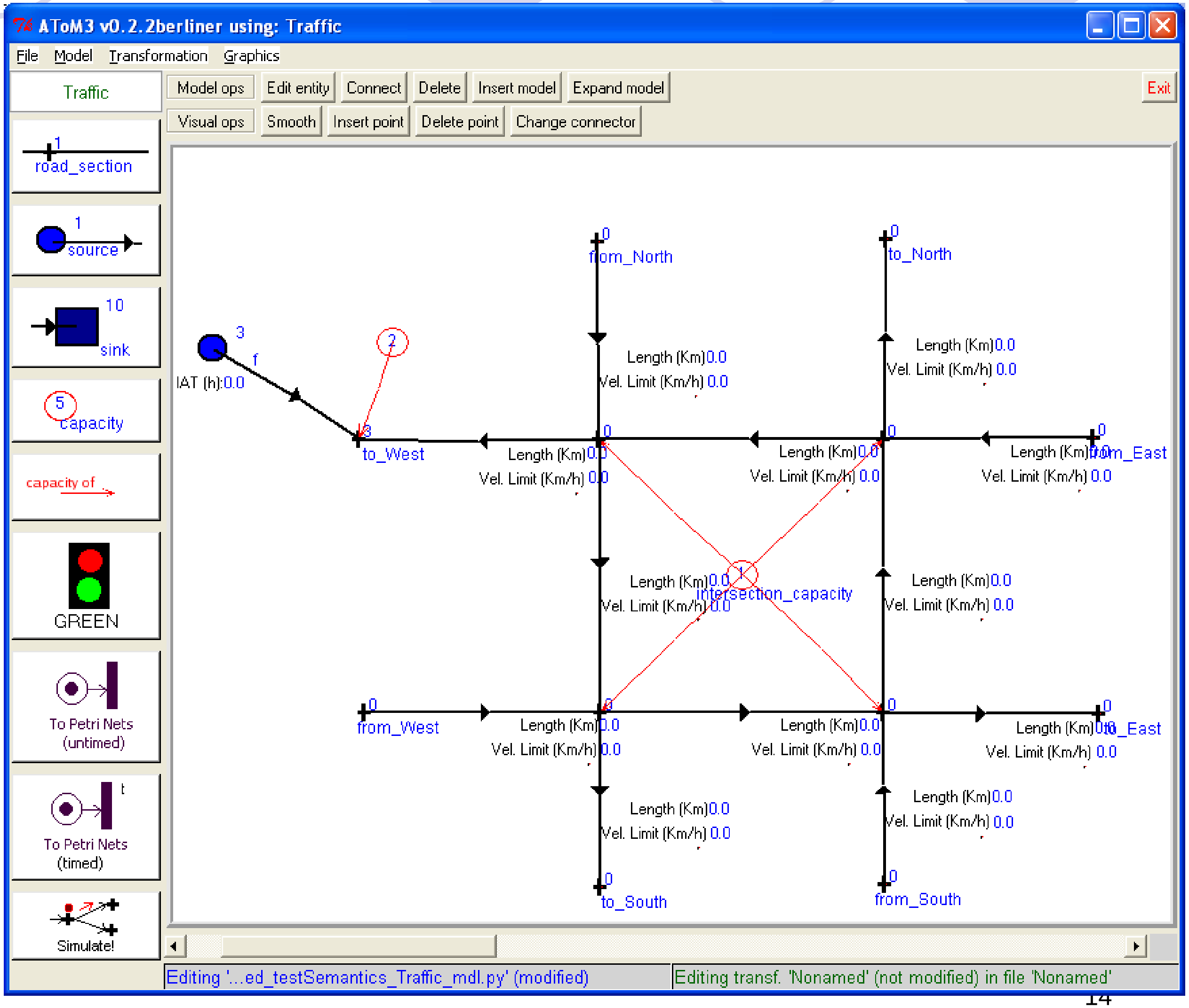
- Constructions that handle the user interface.
- Similar to state machines.
- Concepts are easy to identify.



Smartphone applications / Python

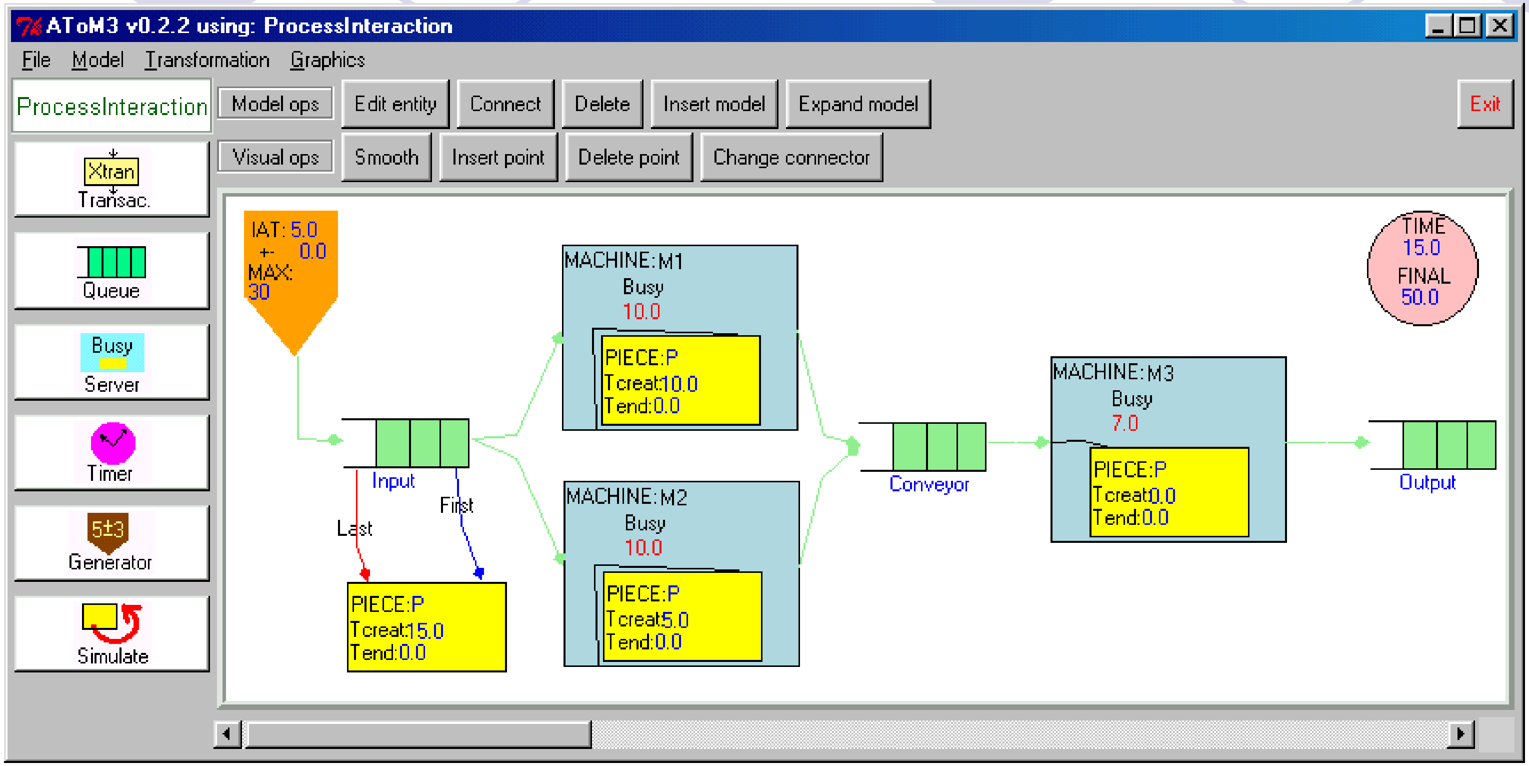
Examples

- Description of physical systems (nets of roads).
- Operational semantics (simulator).
- Denotational semantics (transformation into Petri nets).



Road nets / Petri nets

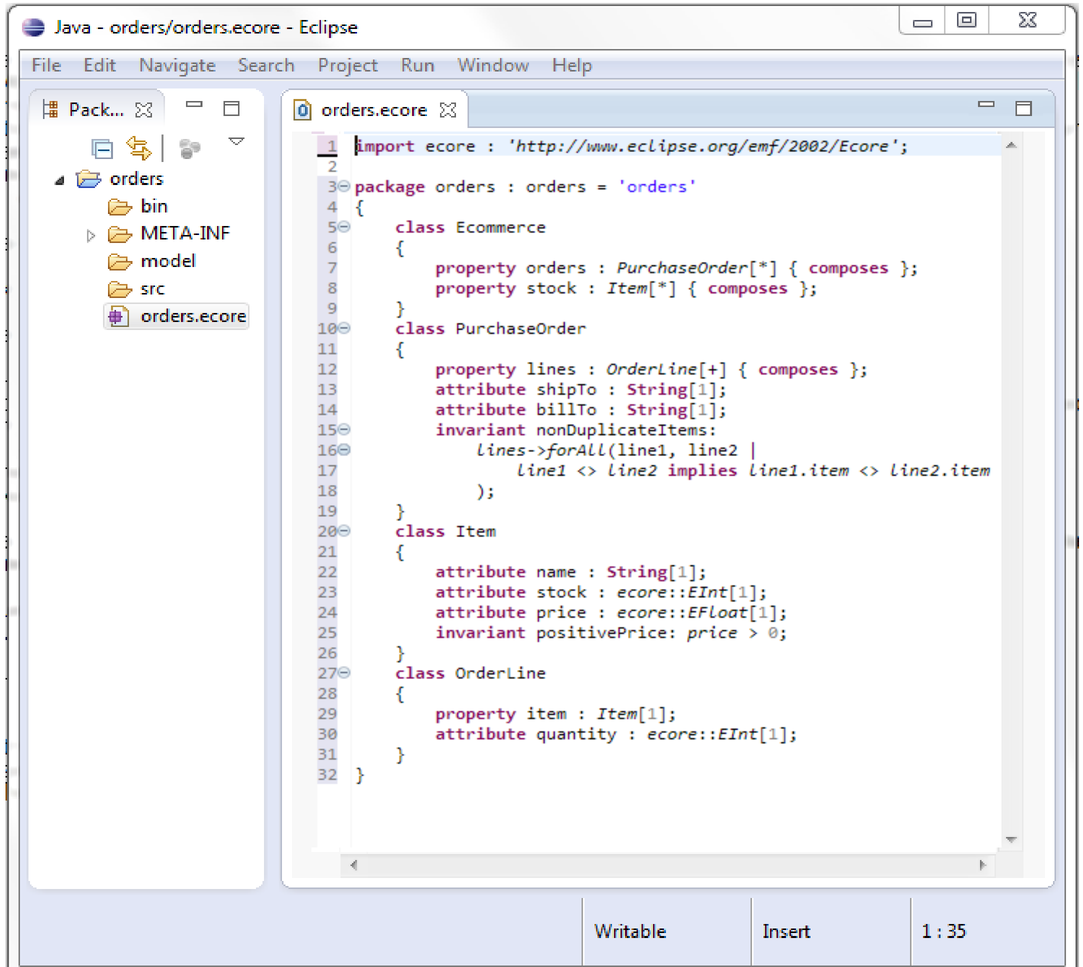
Examples



- DSVL to describe manufacture systems (discrete simulation).
- Educational purpose.

Examples

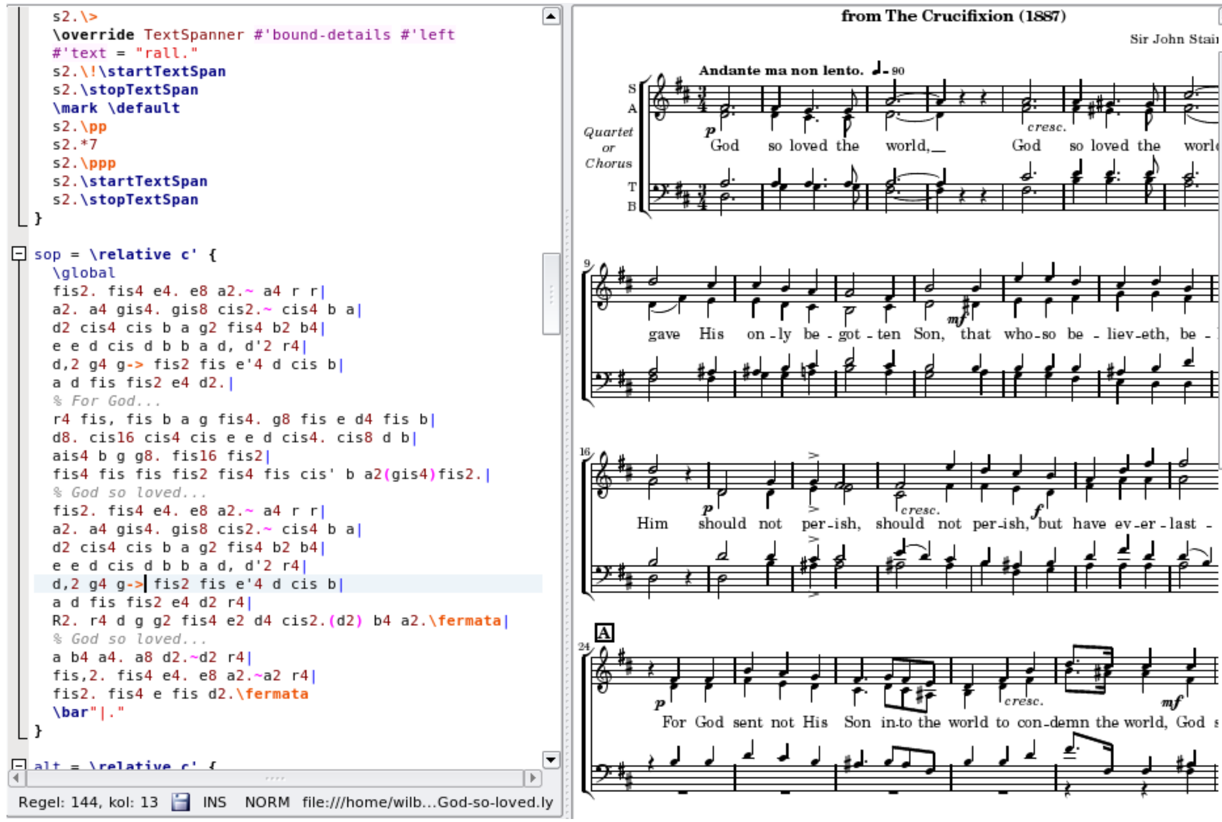
- OclInEcore: textual DSL to specify meta-models.
- Java code generation.



meta-model + constraints / Java code

Examples

- LilyPond: textual DSL to specify music sheets.
- Graphical music sheet generation.



(textual) music sheet / (graphical) music sheet

Index

- Introduction.
- **Types of modelling environments.** 
  - **Free-hand environments.**
  - **Syntax-directed environments.**
- Technologies to build DSLs.
- Bibliography.

Free-hand environments

- “Low-level” editors which allow users to manipulate directly the diagram.
- Parser to recognise the syntactic structure and correctness of the diagram.
- Freedom in the way diagrams are edited.
- This can be a disadvantage, as users have no guidance on how to build their models.

Syntax-directed environments

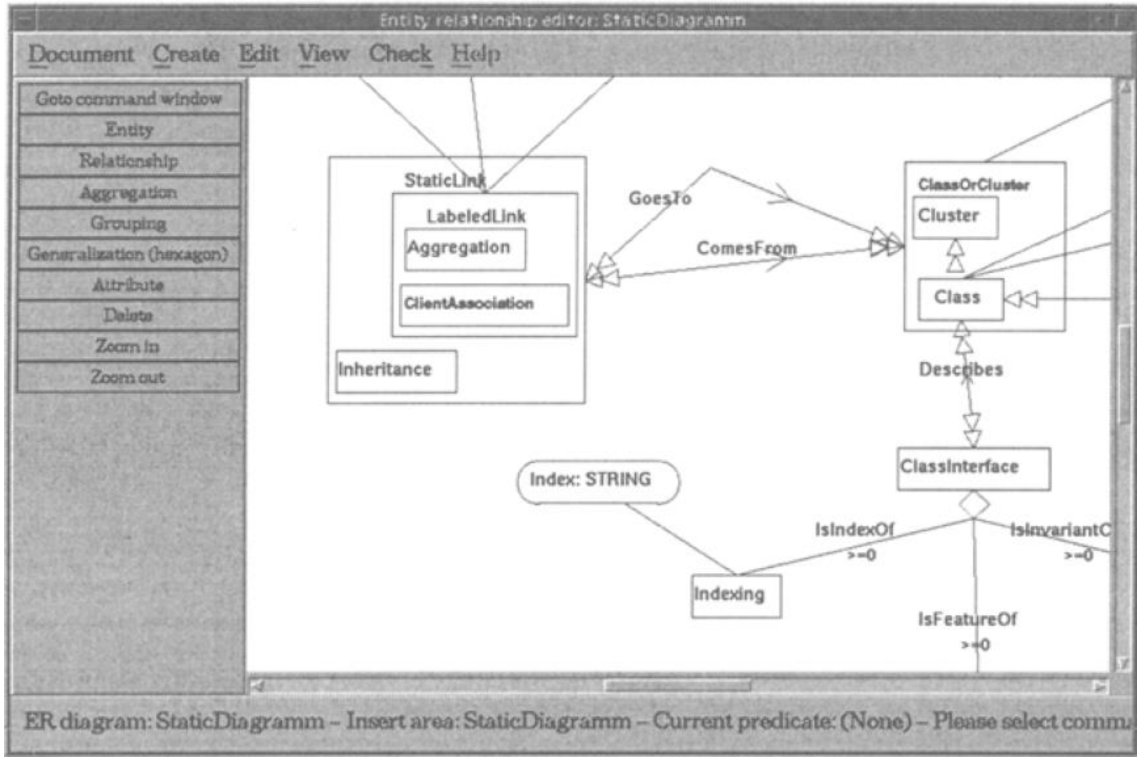
- Editing actions are modelled as graph grammar rules.
- It requires both creation rules and deleting rules.
- Interesting technique for complex editing actions (e.g. creating or connecting many elements).
- Having many different rules can make this approach difficult to manage.

Index

- Introduction.
- Types of modelling environments.
- **Technologies to build DSLs.** 
- Bibliography.

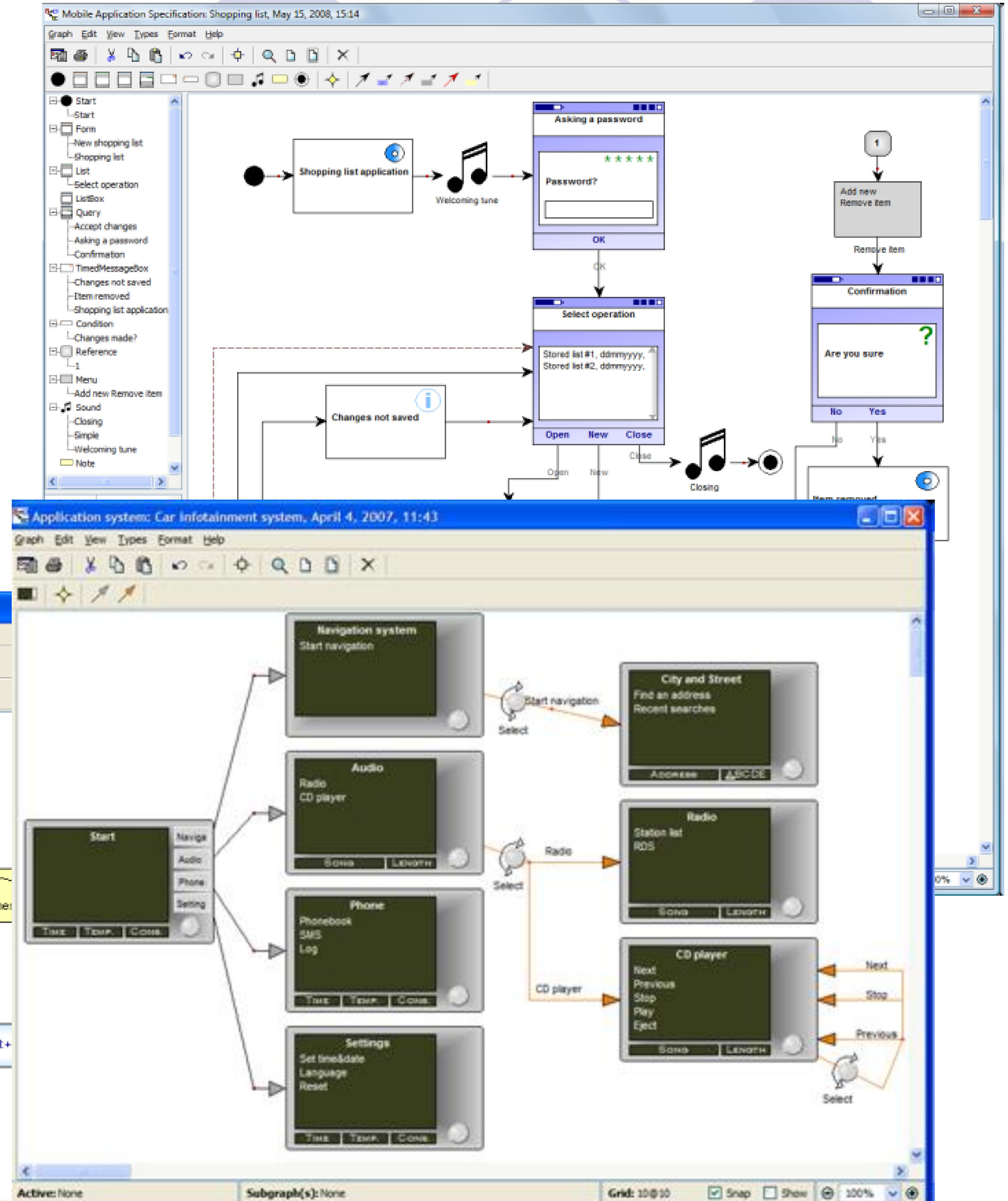
KOGGE

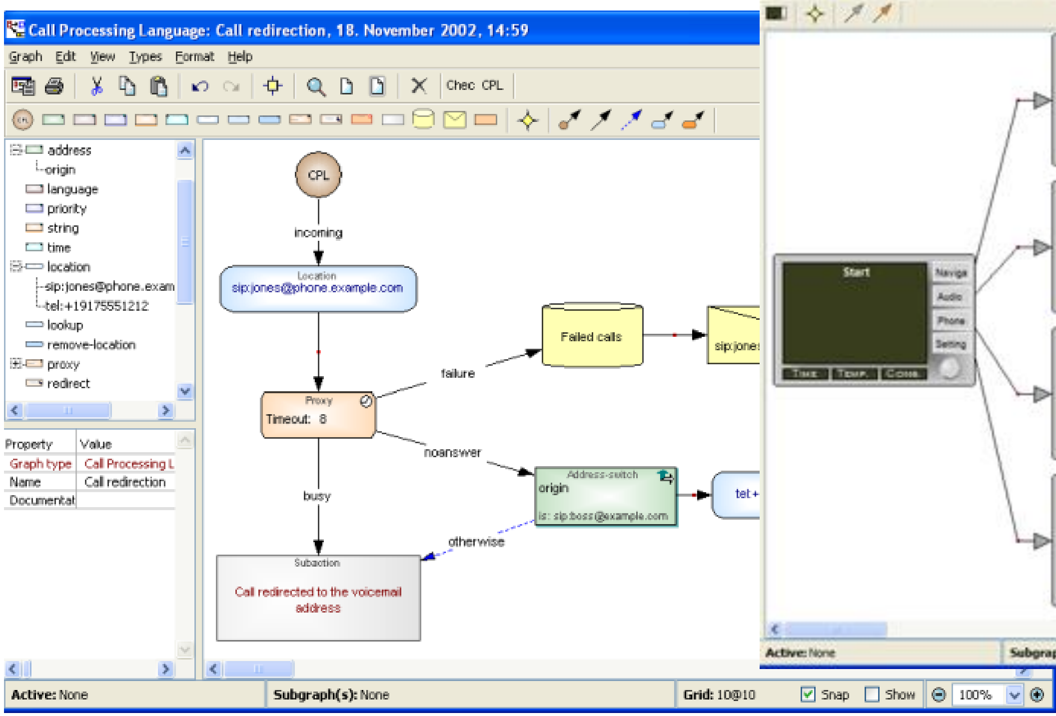
- 1997.
- Ebbert, Süttenbach, Uhe (Loblenz).
- Meta-CASE tools, to build CASE tools.



MetaEdit+

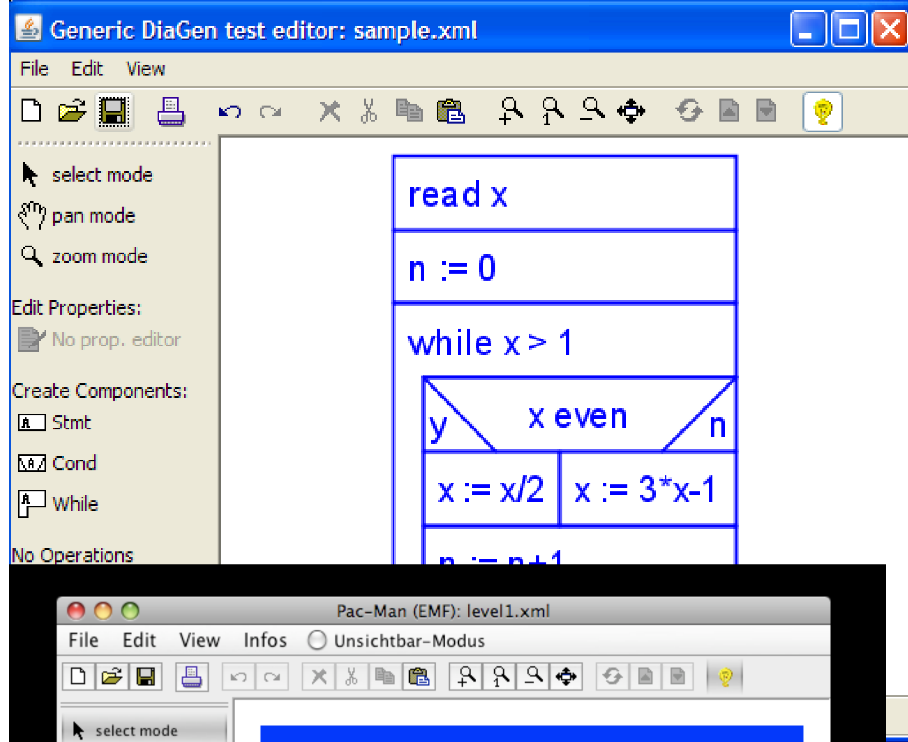
- First version in 1995 (http://www.metacase.com).
- Commercial, multi-user.

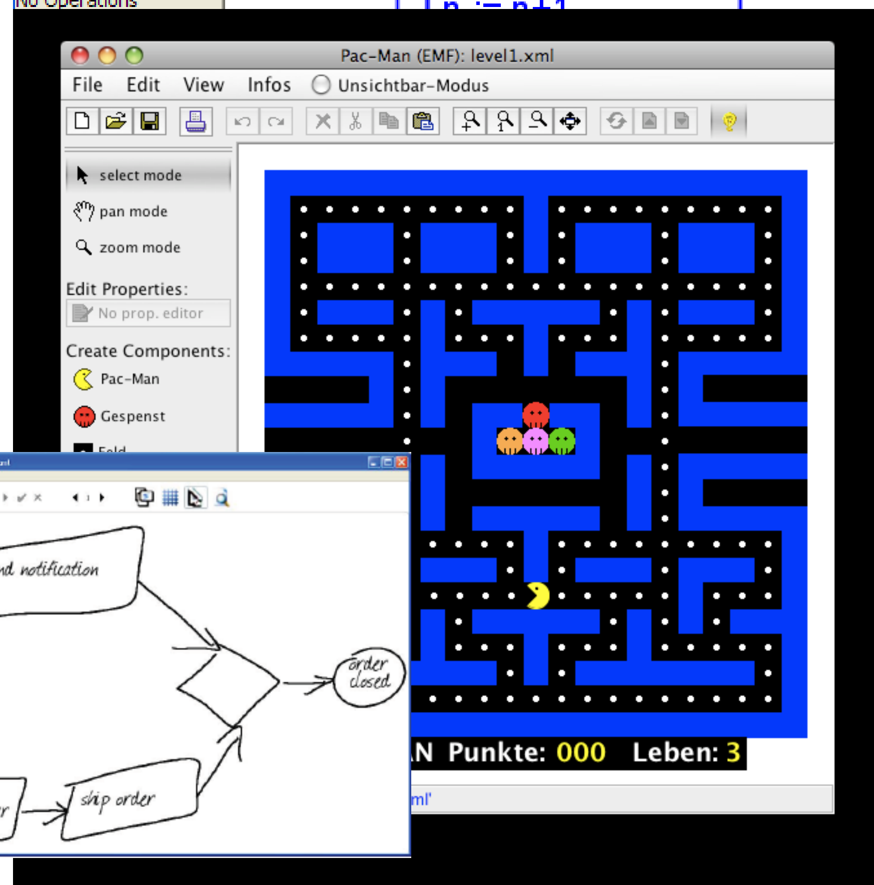


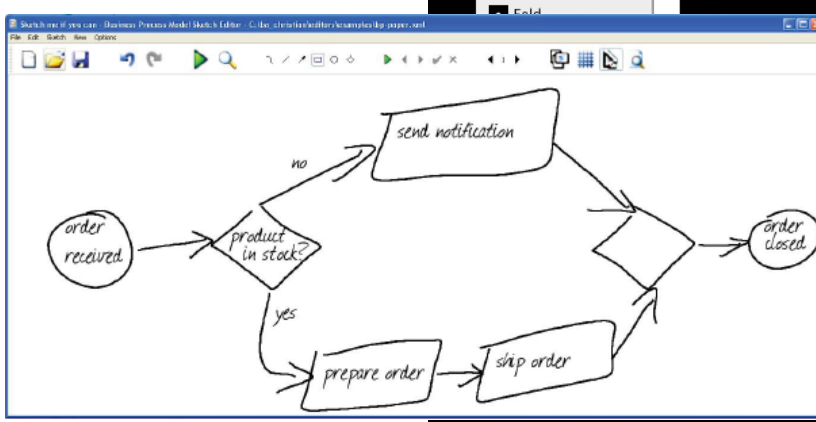


DiaGen/DiaMeta

- First version in 1993 (http://www.unibw.de/inf2/DiaGen/).
- Based on hypergraph grammars.
- Sketching.
- Mark Minas (Munich).

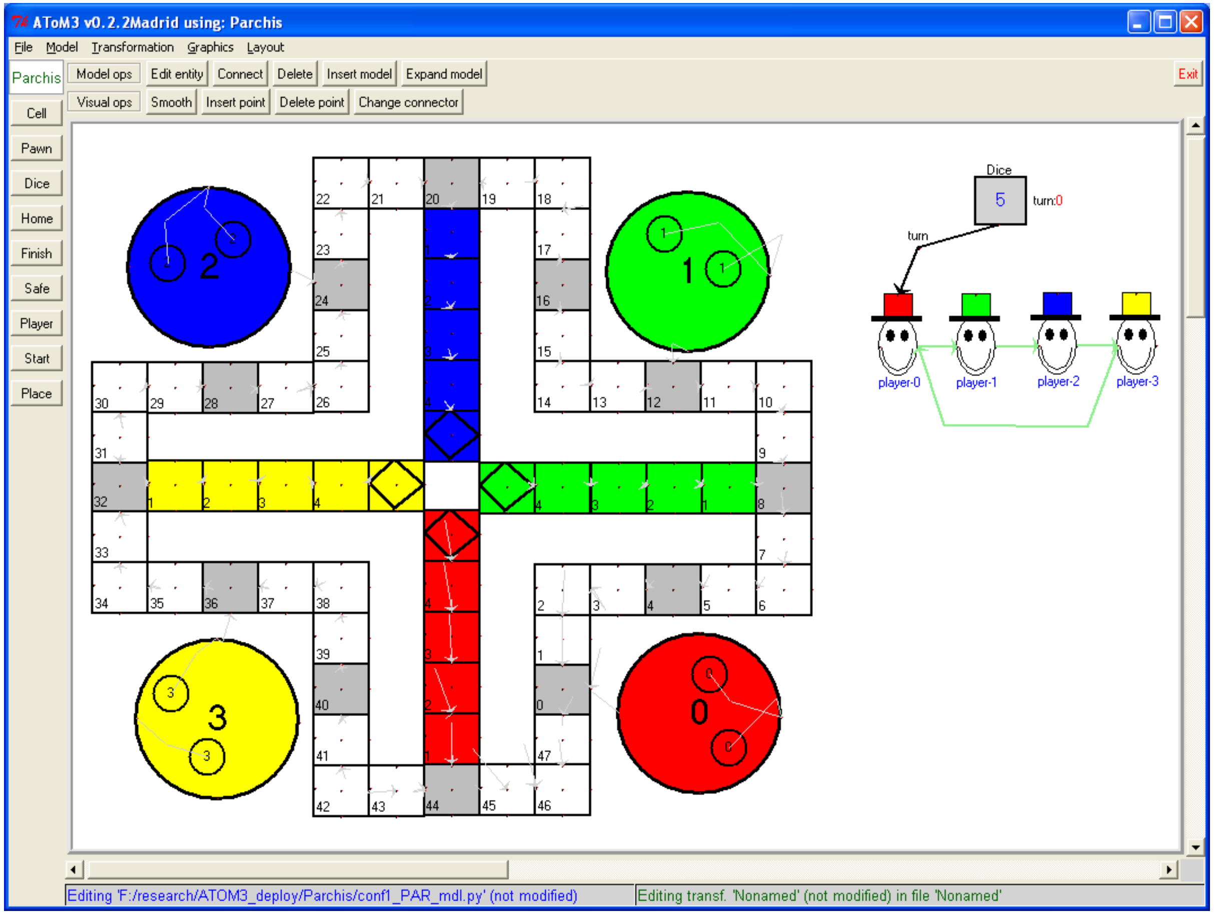






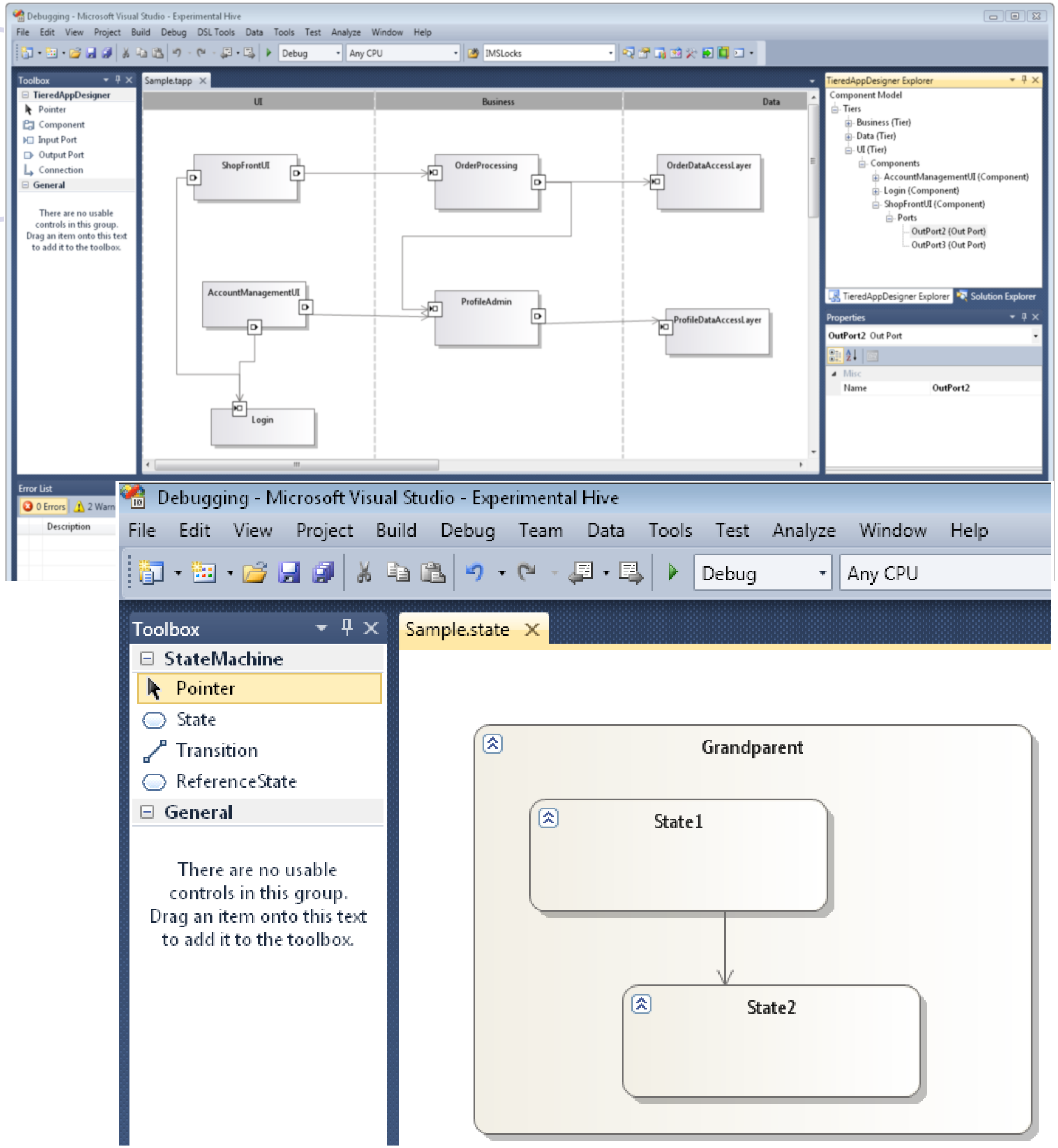
AToM^3

- 2002.
- Model manipulation can be graphically defined using graph transformation.
- Simulation.



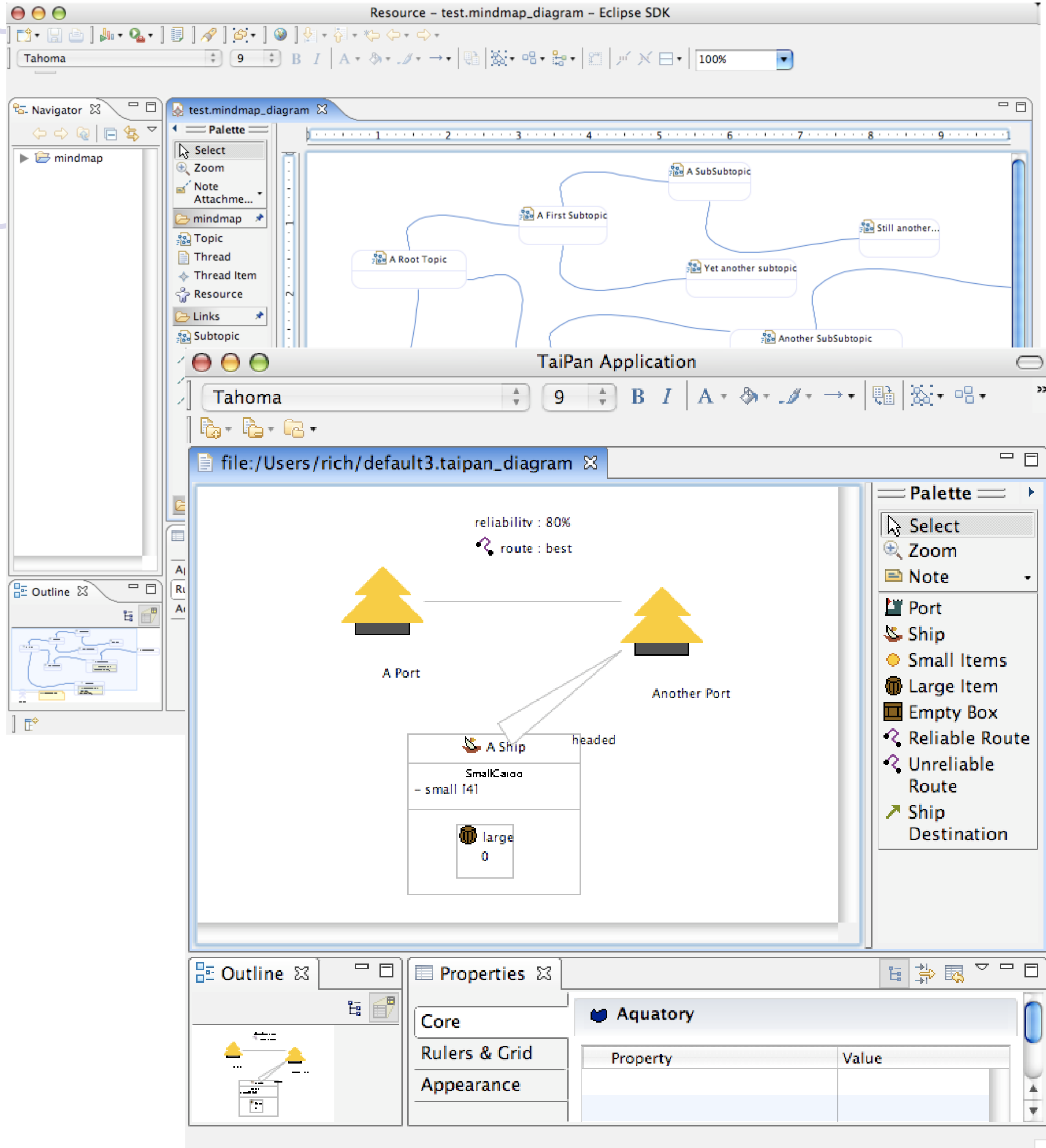
DSL Tools

- Microsoft/ Visual Studio.



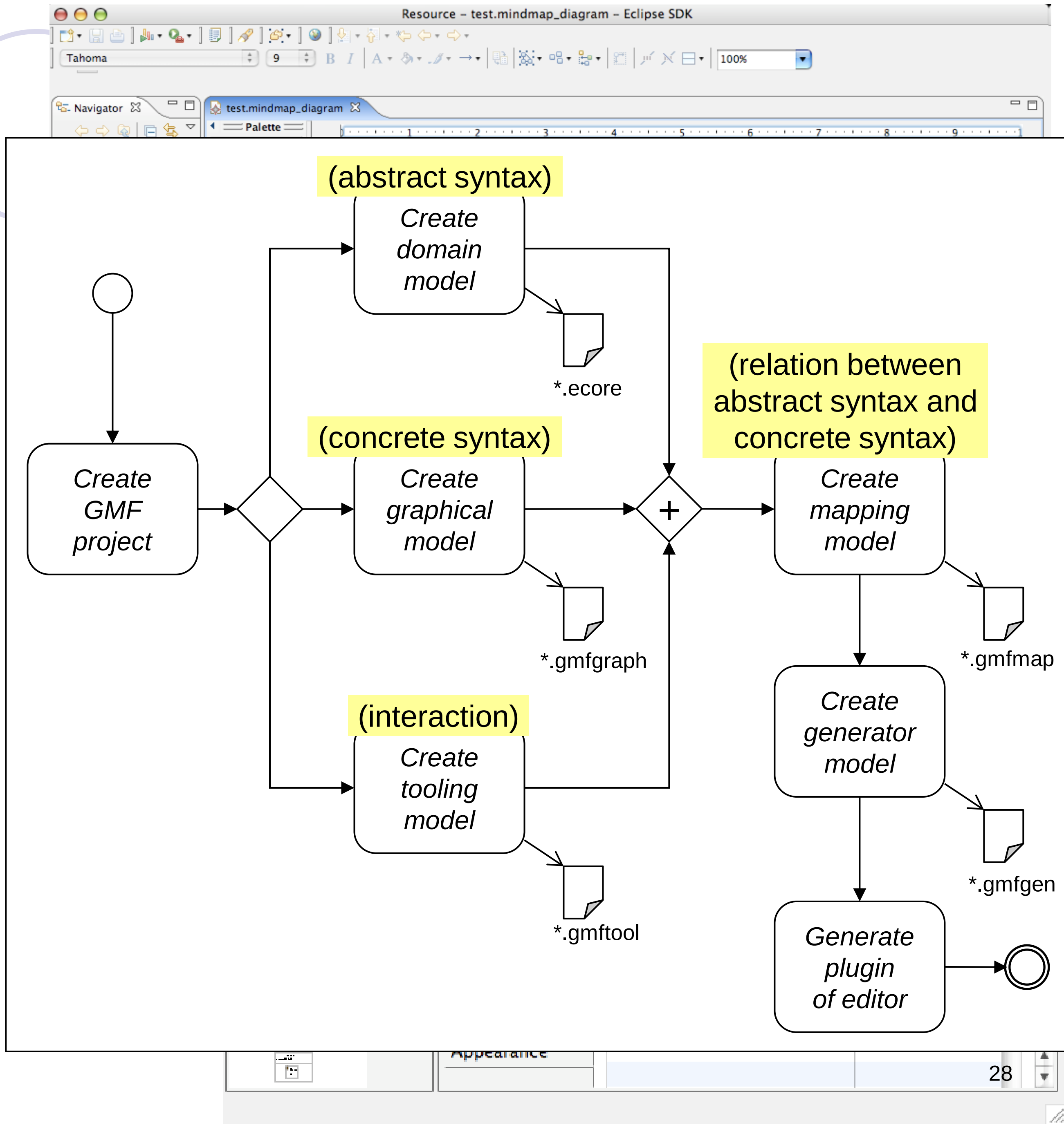
GMF

- EMF/Eclipse.



GMF

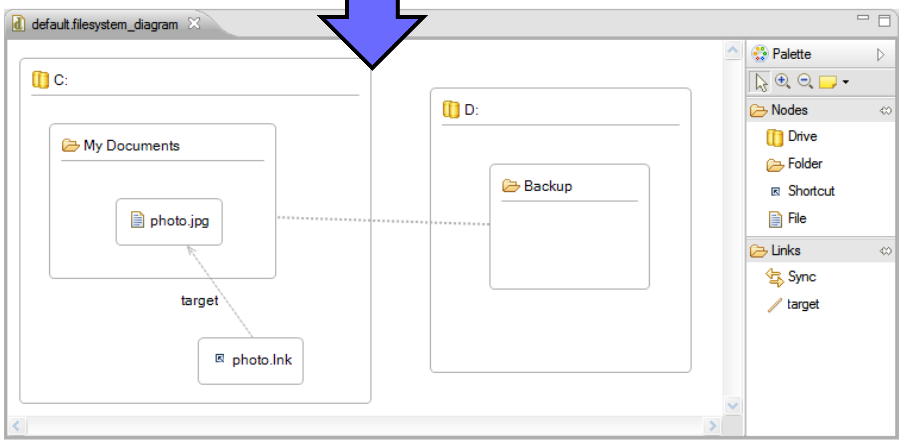
- EMF/Eclipse.
- Complex!



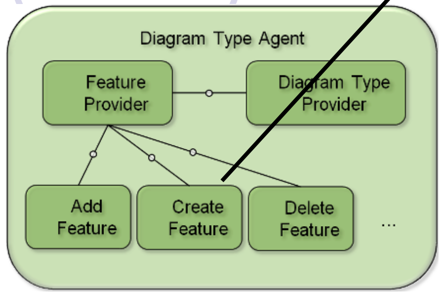
Eugenia

- EMF/Eclipse.
- http://www.eclipse.org/gmt/epsilon/doc/articles/eugenia-gmf-tutorial/.
- It generates GMF editors from annotated ecore meta-models.
- The generated GMF editor must be maintained by hand.

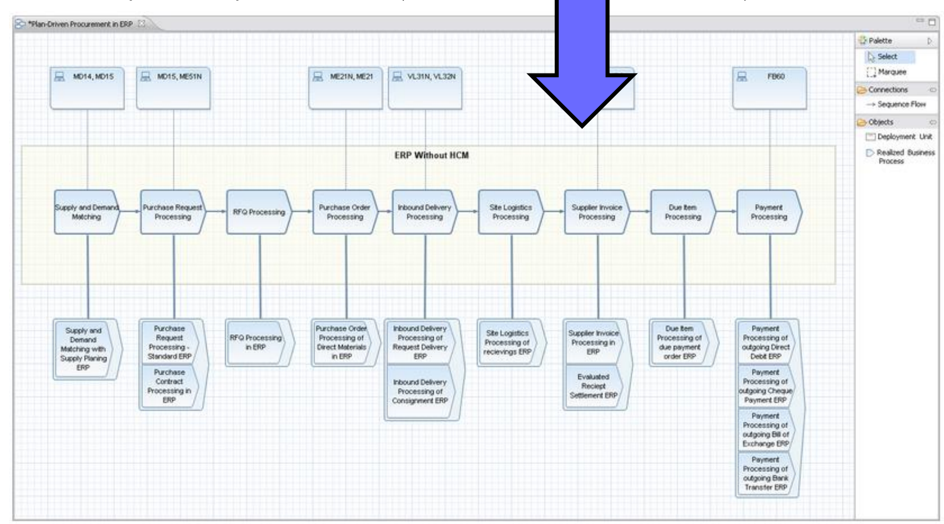
```
class Folder extends File {
    @gmf.compartment
    val File[*] contents;
}

class Shortcut extends File {
    @gmf.link(target.decoration="arrow", style="dash")
    ref File target;
}

@gmf.link(source="source", target="target", style="dot", width="2")
class Sync {
    ref File source;
    ref File target;
}

@gmf.node(label = "name")
class File {
    attr String name;
}
```



Graphiti



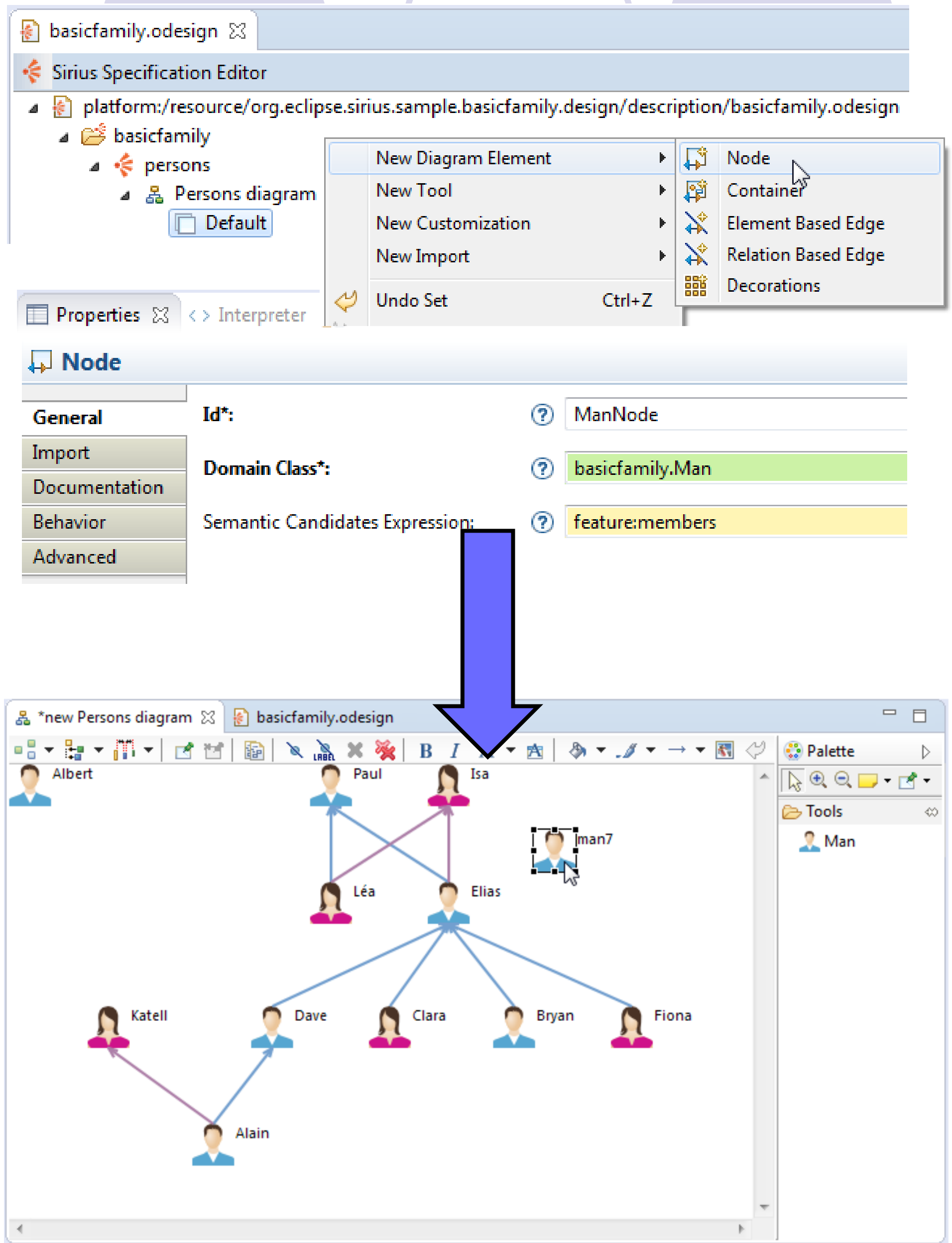

```
public class CreatePurchaseOrderFeature
    extends AbstractCreateFeature
    implements ICreateFeature {

    public CreatePurchaseOrderFeature(IFeatureProvider fp) {
        super(fp, "PurchaseOrder", "Creates a new PurchaseOrder");
    }

    @Override
    public boolean canCreate(ICreateContext context) {
        // check appropriate context
        return context.getTargetContainer() instanceof Diagram;
    }

    @Override
    public Object[] create(ICreateContext context) {
        // create the domain object
        PurchaseOrder newPurchaseOrder =
          OrdersFactory.eINSTANCE.createPurchaseOrder();

        // attribute values
        String shipTo = (String) JOptionPane.showInputDialog
          (new JFrame(), “Ship to”);
        newPurchaseOrder.setShipTo(shipTo);

        // add object to diagram
        getDiagram().eResource().getContents().add(newPurchaseOrder);

        // add graphical representation of obj
        addGraphicalRepresentation(context, newPurchaseOrder);
}
```



- EMF/Eclipse.
- http://www.eclipse.org/graphiti/.
- Flat learning curve (Java API + Graphiti objects), high flexibility, common look and feel with sensible defaults.
- Spray (https://code.google.com/a/eclipselabs.org/p/spray/): DSL to describe Graphiti editors.

Sirius

- EMF/Eclipse.
- http://www.eclipse.org/sirius/.
- Tutorials: http://www.eclipse.org/sirius/getstarted.html.
- Easy to use; interpreted at runtime; definition is a model describing syntax, editing tools and validation rules.



Index

- Introduction.
- Types of modelling environments.
- Technologies to build DSLs.
- **Bibliography.** 

Bibliography

- Domain-specific languages:
  - OOPSLA workshops on Domain Specific Languages.
  - "*Defining domain-specific modeling languages: Collected experience*". 2004. J. Luoma, S. Kelly, J.-P. Tolvanen. OOPSLA Workshop on Domain Specific Languages.
- Visual languages:
  - Conference GT-VMT "Graph Transformation Visual Modelling Techniques".
  - Conference IEEE VL/HCC "Visual Language / Human Centric Computing".