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.**
- Generation of modelling environments.
- Tools.
- 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 general-purpose languages.
- DSLs are extensively created/used in MDE solutions.

Problem domain vs Solution domain

Domain Specific Languages

problem space

- smaller semantic gap
- oriented to the user way of thinking

(semi-)automatic transformation

General Purpose Languages

solution space

- need to transform into technical domain
- oriented to the developer way of thinking

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

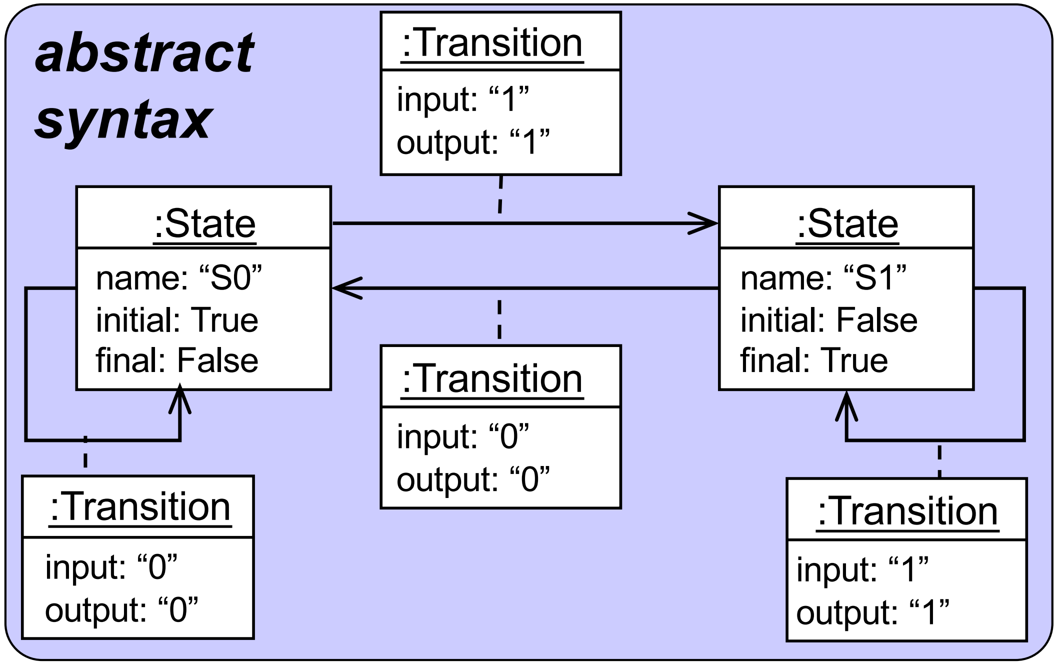
- **External**: they are built from scratch.
  - Flexibility on the concrete syntax of the language
  - Costly implementation (requires implementing parser, syntactic analyzer, interpreter or compiler, editing environment, etc.)
  - …but there are frameworks that facilitate their development, like Sirius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

External Domain-Specific Languages

- DSLs can be graphical, textual, or a combination of text and graphics. For instance:
  - 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).
  - Spatial constraint languages (e.g. QOCA, https://www.swmath.org/software/756)



abstract syntax

:State
name: "S0"
initial: True
final: False

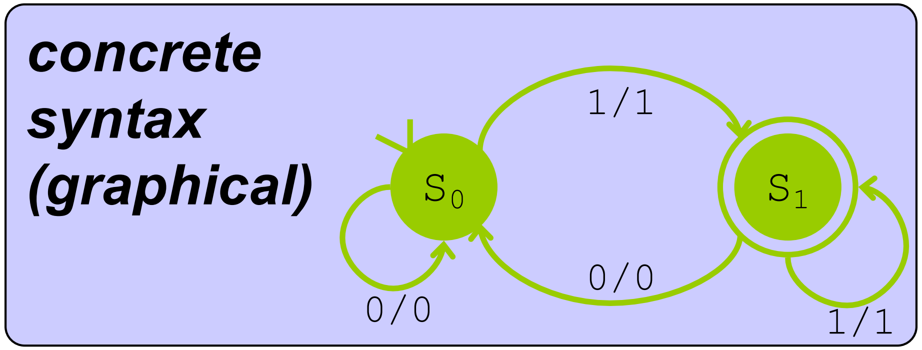
:State
name: "S1"
initial: False
final: True

:Transition
input: "1"
output: "1"

:Transition
input: "0"
output: "0"

:Transition
input: "0"
output: "0"

:Transition
input: "1"
output: "1"



concrete syntax (graphical)

concrete syntax (textual)

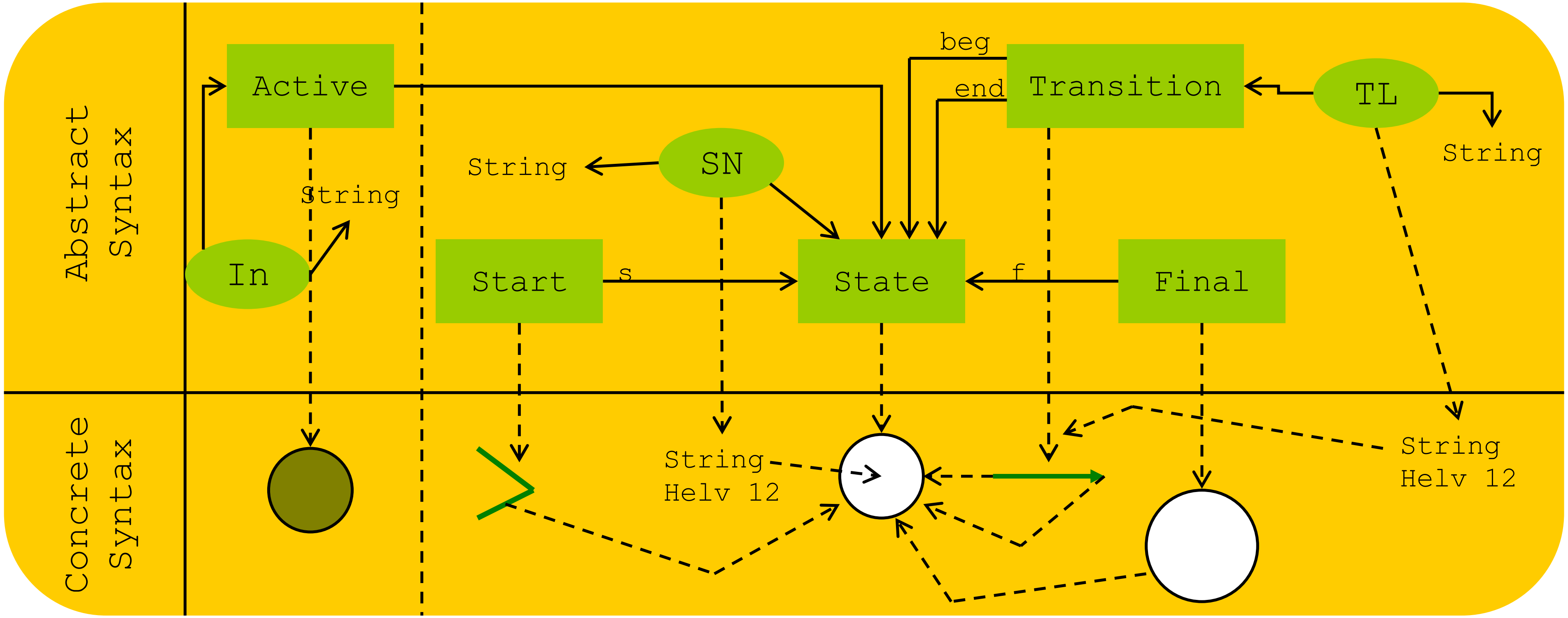
States:
S0 {initial},
S1 {final}

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

Concrete syntax

Creation grammars

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



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 another one for the abstract syntax.

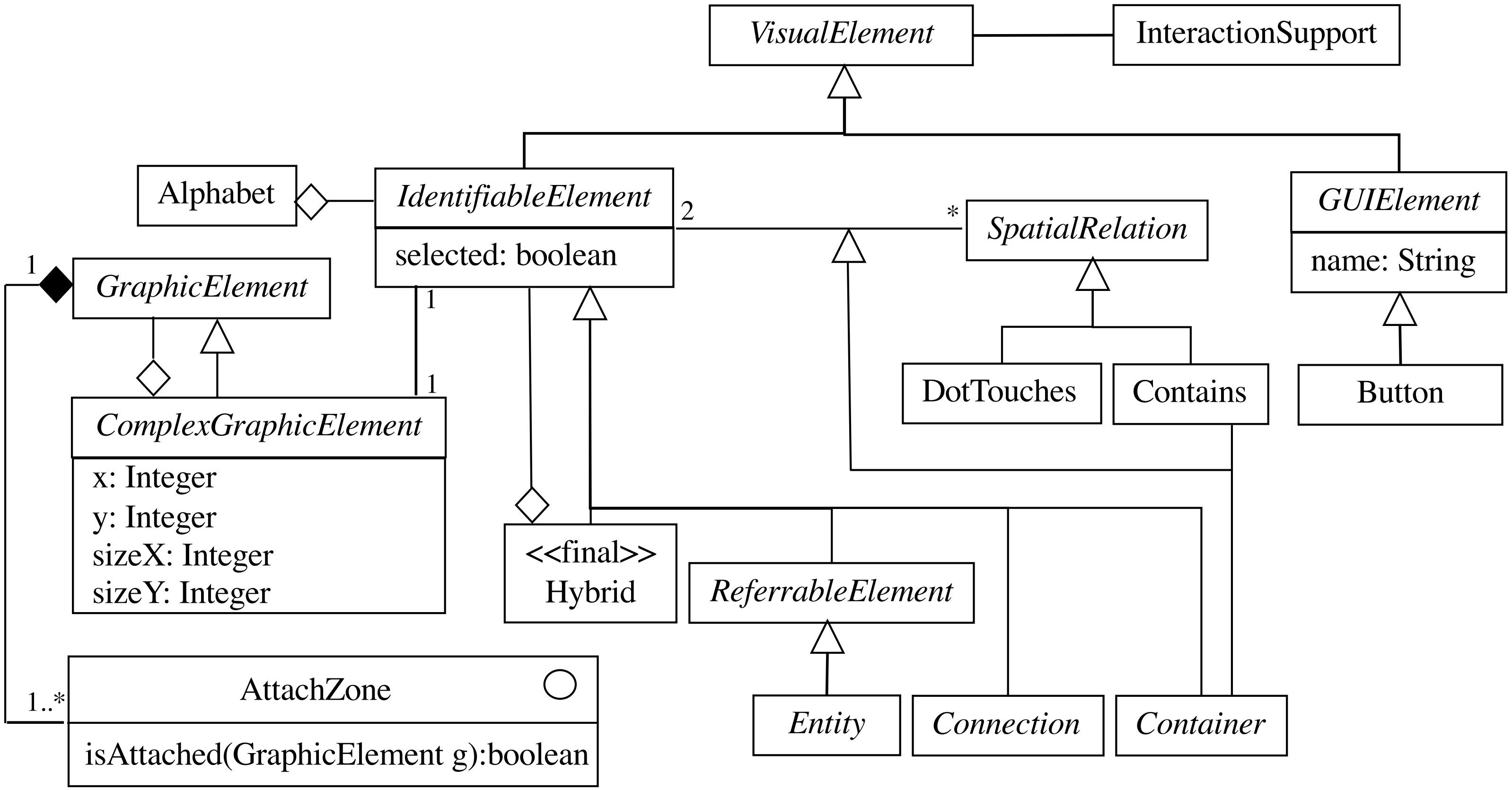
- Spatial relations, e.g., “contained”.

Concrete syntax

Meta-modelling

- We can include spatial relations in the meta-model of the concrete syntax (contained, touches, aligned-with, …).

- The classes of the language meta-model subclasify the base classes.

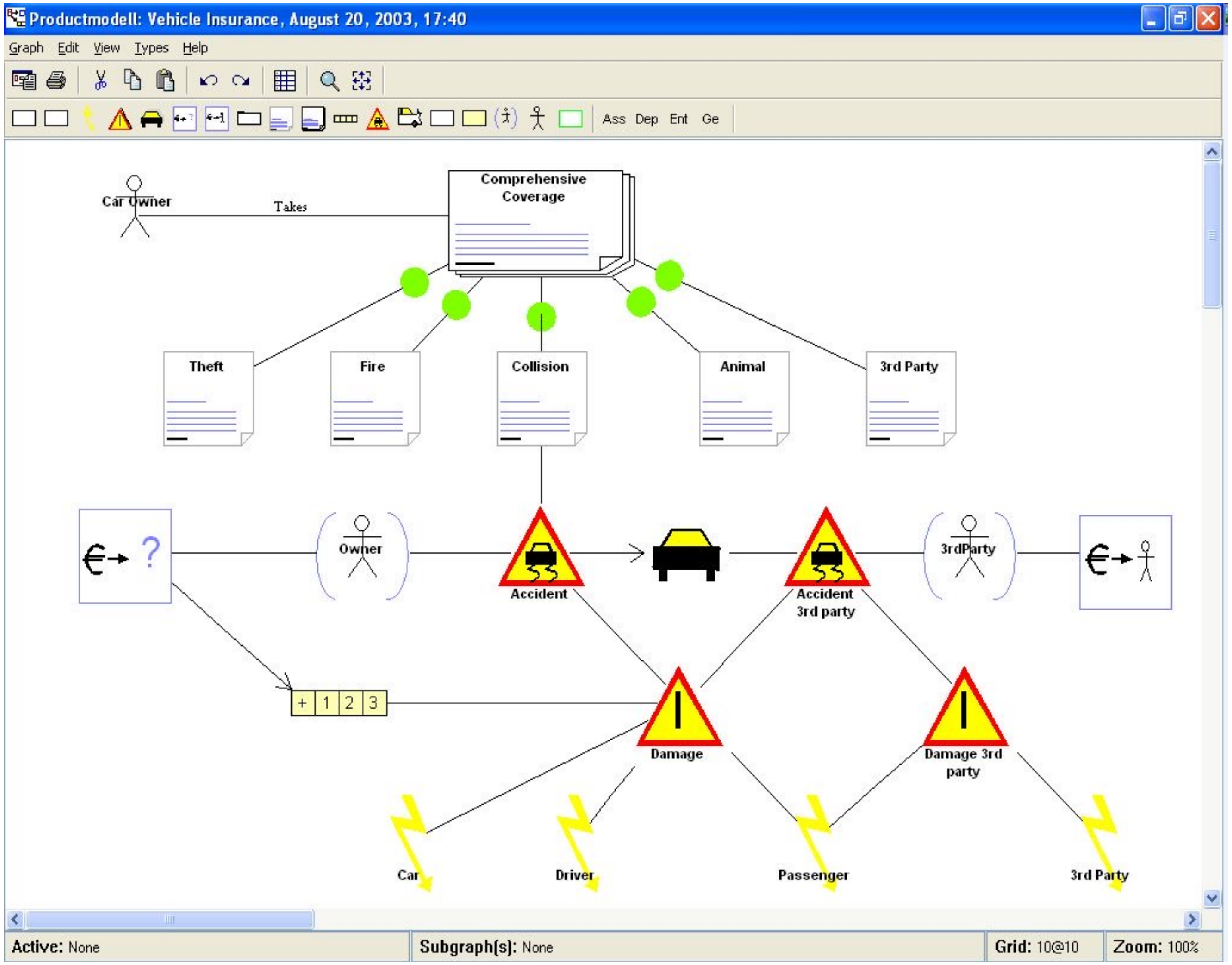


Semantics

- **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

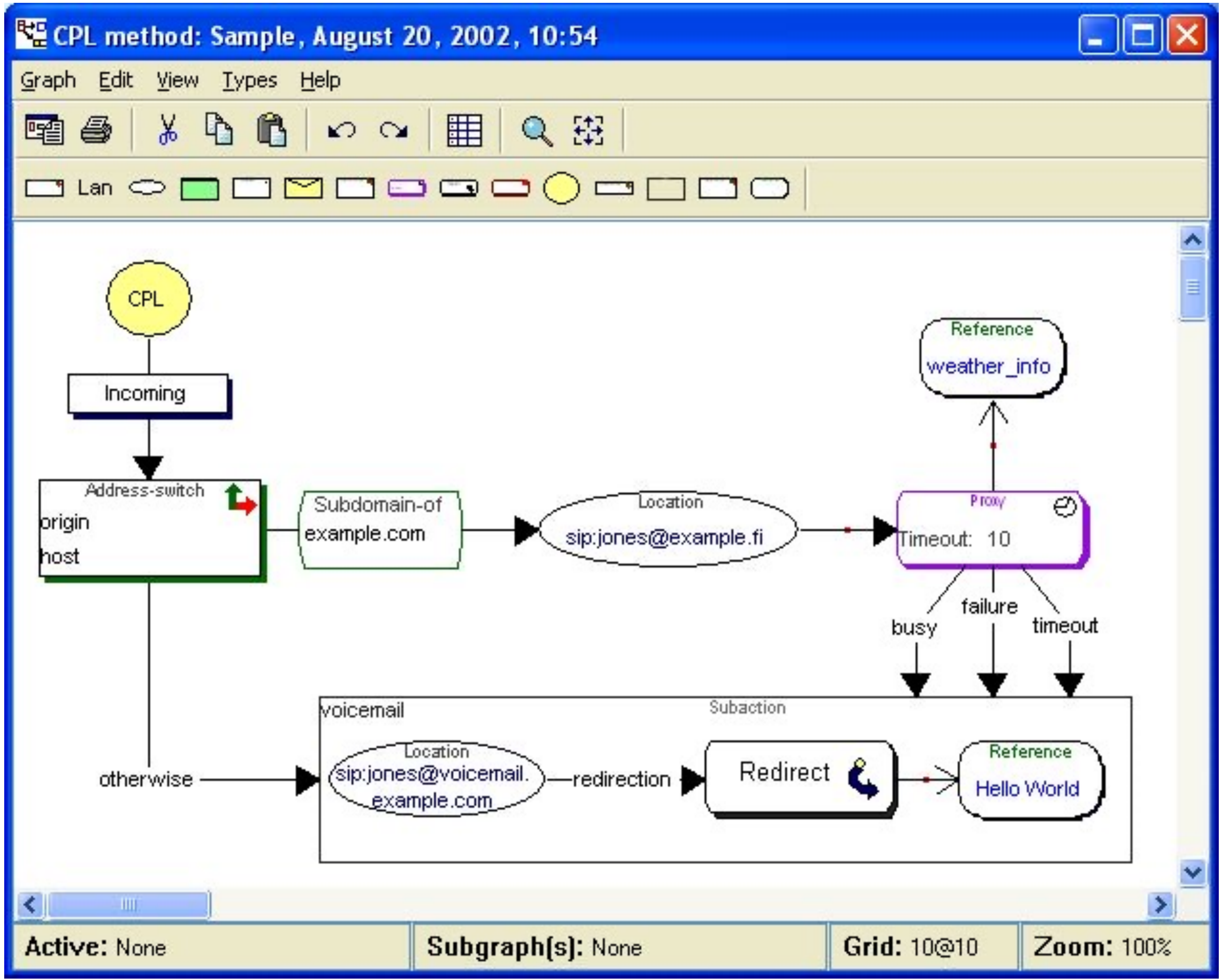
- Expert domain concepts.
- Simple code generation.
- Valid in well-known domains.
- Usable by non-programmers.



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.



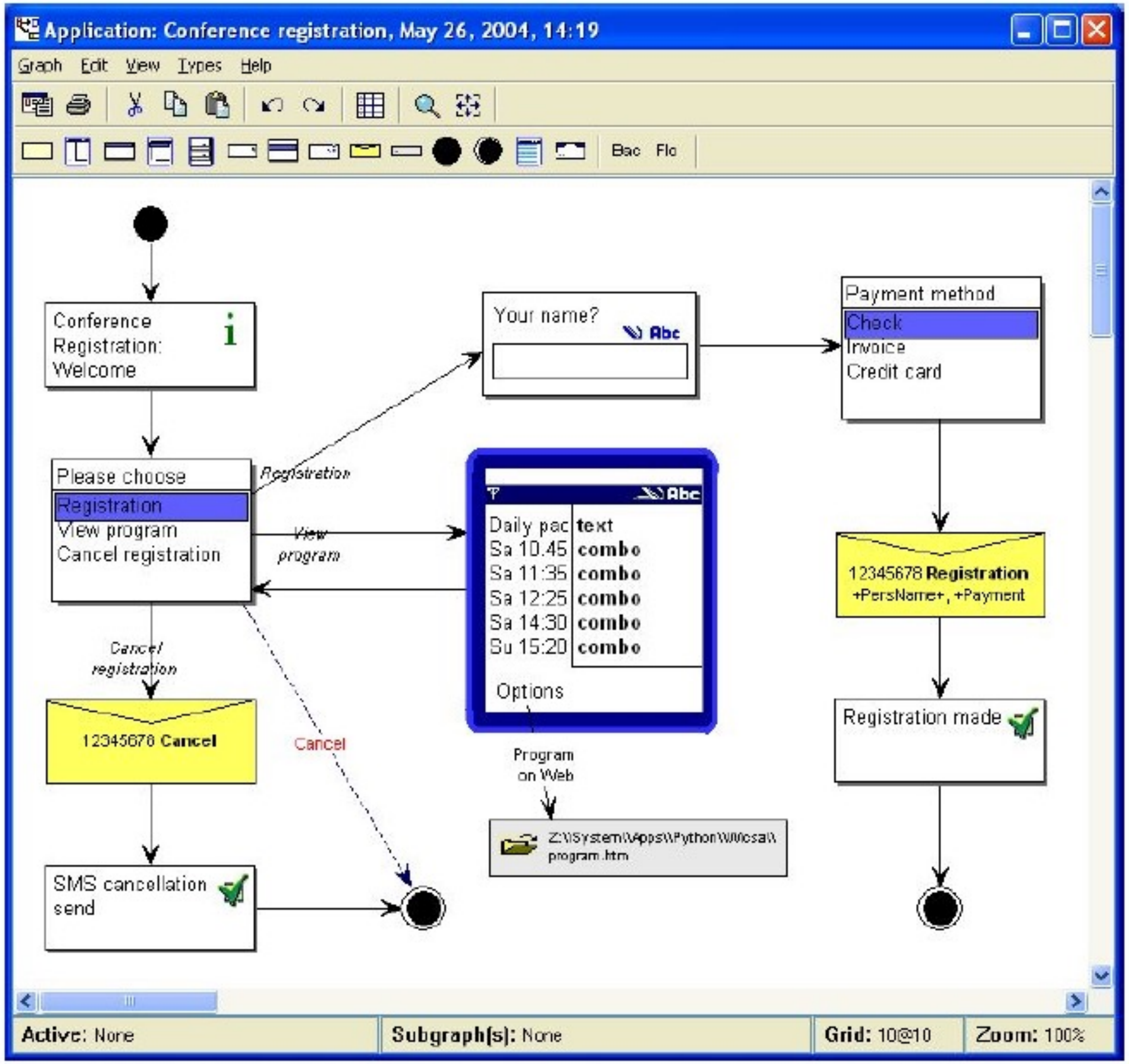
CPL method: Sample, August 20, 2002, 10:54

Active: None | Subgraph(s): None | Grid: 10@10 | Zoom: 100%

Internet Telephony / CPL

Examples

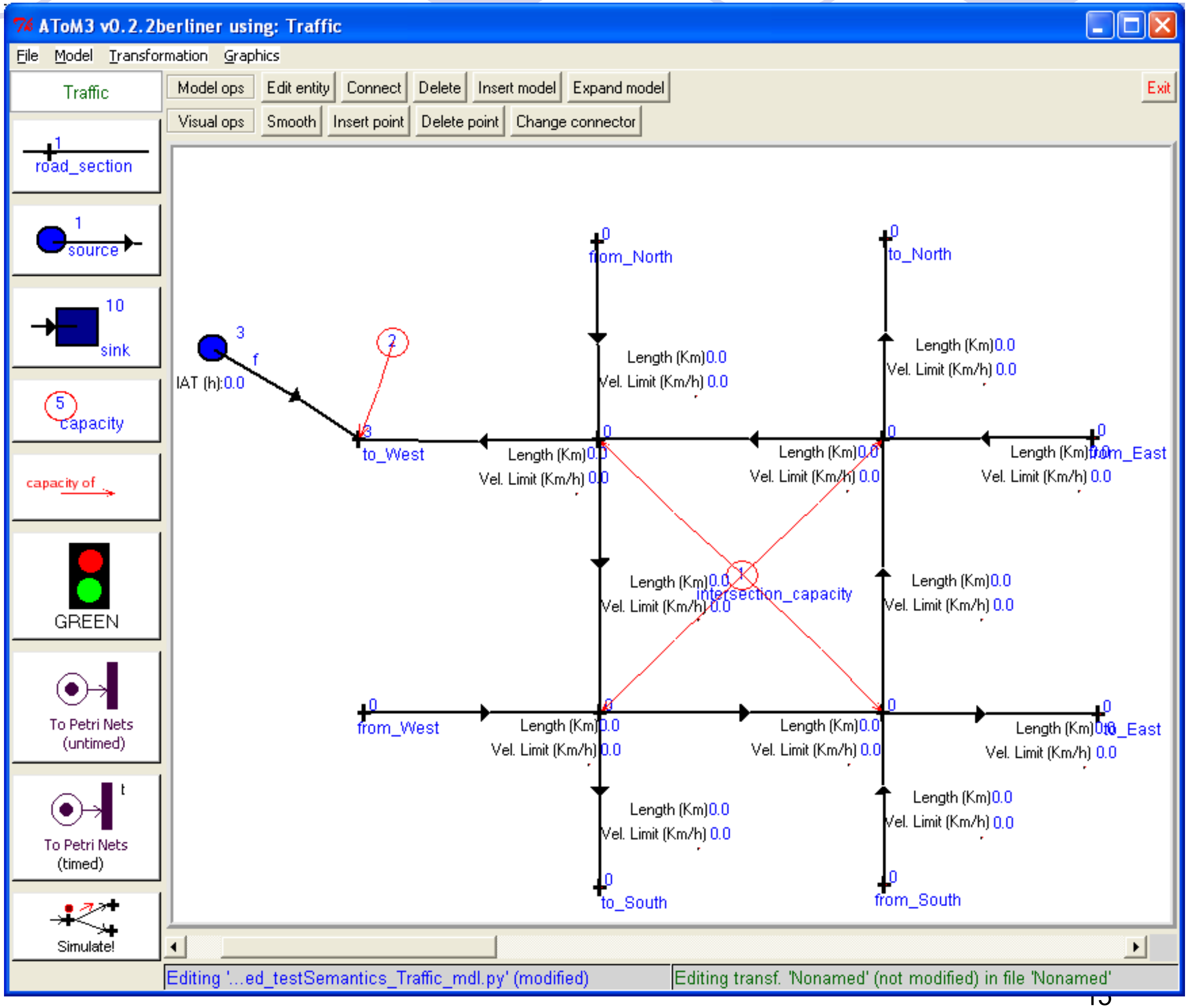
- Constructions that handle the user interface.
- Similar to state machines.
- Concepts are easy to identify.



Smartphone applications / Python

Examples

- DSVL to describe physical systems (nets of roads).
- Transformation into formalisms for analysis (e.g. Petri nets).

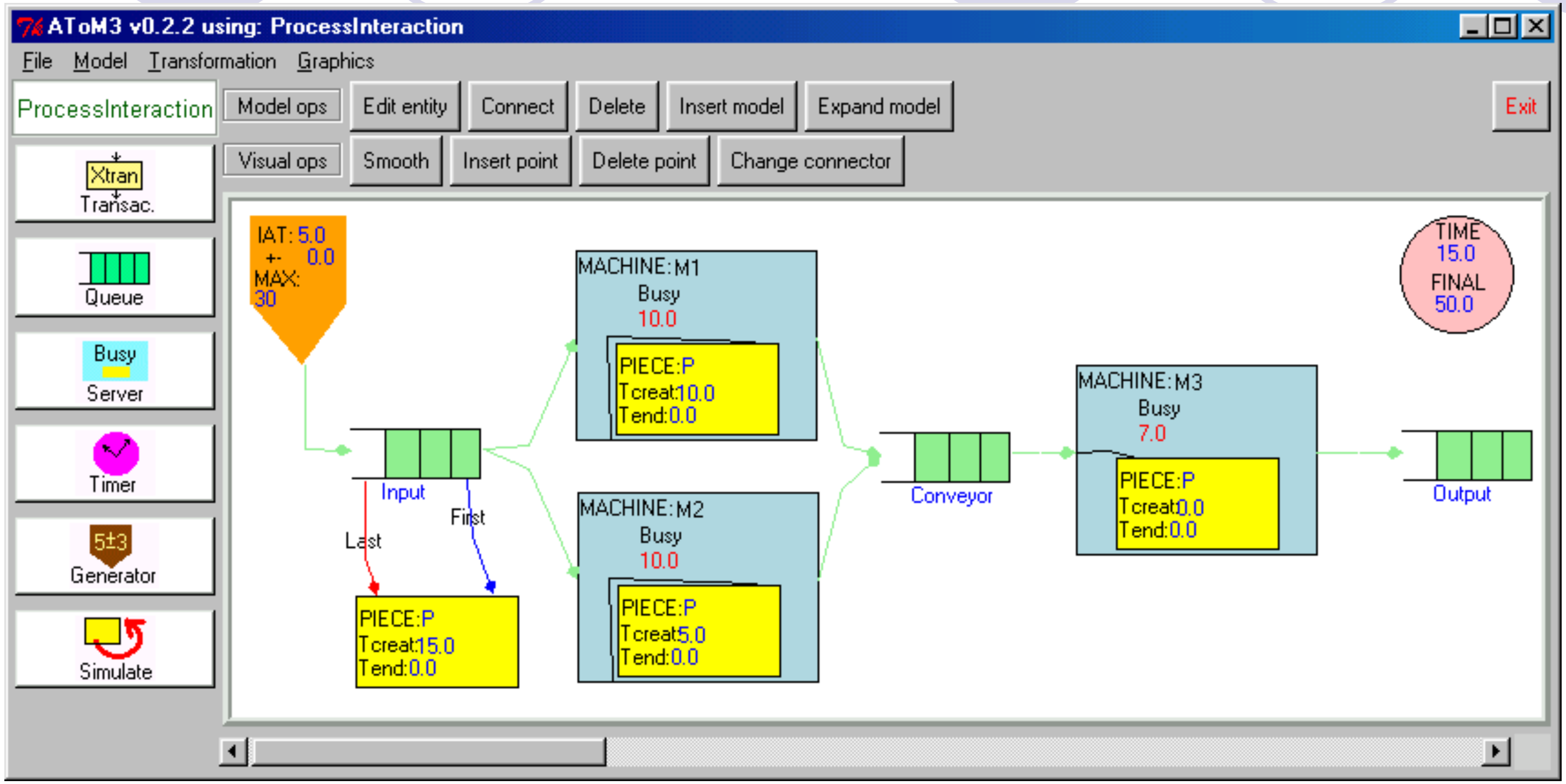


Editing '...ed_testSemantics_Traffic_mdl.py' (modified)

Editing transf. 'Nonamed' (not modified) in file 'Nonamed'

Road nets / Petri nets

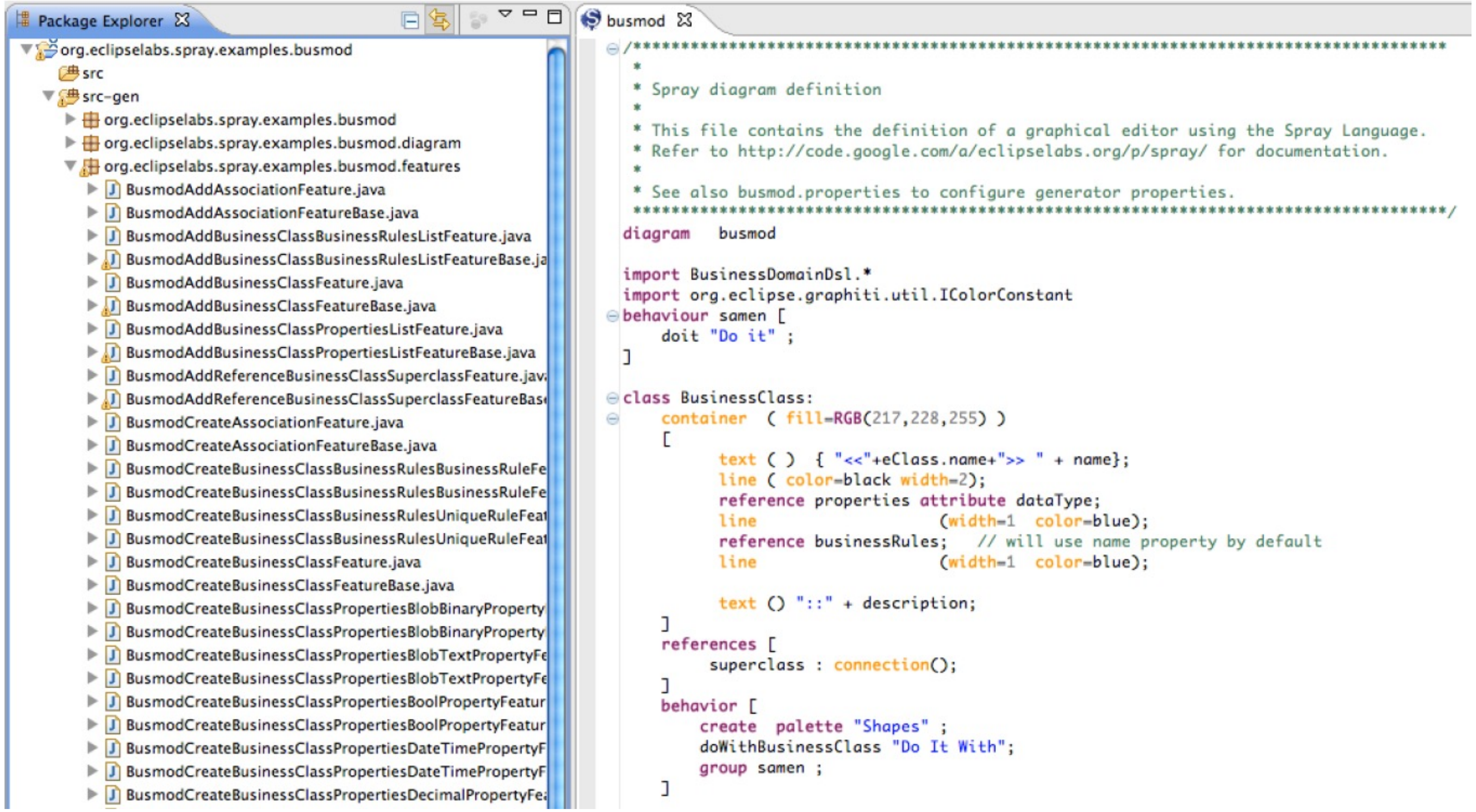
Examples



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

Examples

- Spray: textual DSL to specify Graphiti graphical editors.
- Java code generation.



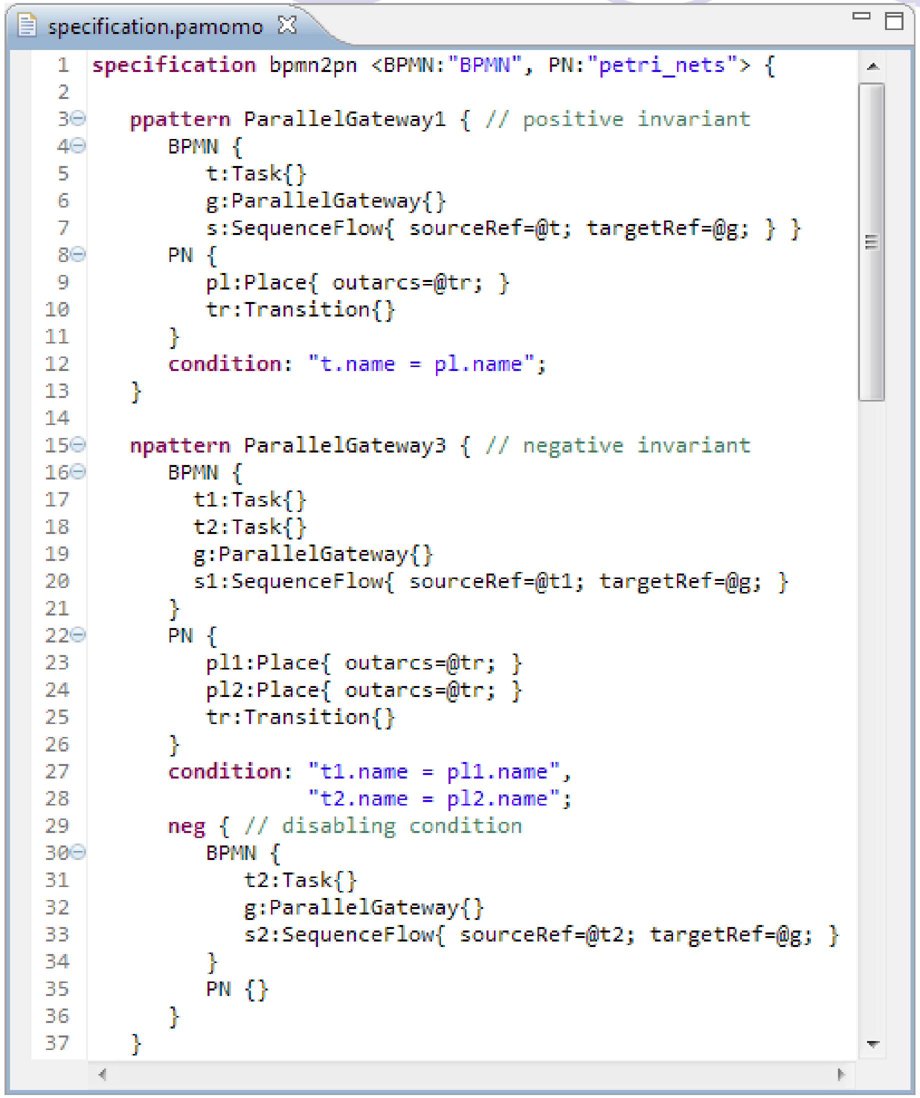
```
org.eclipselabs.spray.examples.busmod
  src
  src-gen
    org.eclipselabs.spray.examples.busmod
    org.eclipselabs.spray.examples.busmod.diagram
    org.eclipselabs.spray.examples.busmod.features
      BusmodAddAssociationFeature.java
      BusmodAddAssociationFeatureBase.java
      BusmodAddBusinessClassBusinessRulesListFeature.java
      BusmodAddBusinessClassBusinessRulesListFeatureBase.ja
      BusmodAddBusinessClassFeature.java
      BusmodAddBusinessClassFeatureBase.java
      BusmodAddBusinessClassPropertiesListFeature.java
      BusmodAddBusinessClassPropertiesListFeatureBase.java
      BusmodAddReferenceBusinessClassSuperclassFeature.java
      BusmodAddReferenceBusinessClassSuperclassFeatureBase
      BusmodCreateAssociationFeature.java
      BusmodCreateAssociationFeatureBase.java
      BusmodCreateBusinessClassBusinessRulesBusinessRuleFe
      BusmodCreateBusinessClassBusinessRulesBusinessRuleFe
      BusmodCreateBusinessClassBusinessRulesUniqueRuleFeat
      BusmodCreateBusinessClassBusinessRulesUniqueRuleFeat
      BusmodCreateBusinessClassFeature.java
      BusmodCreateBusinessClassFeatureBase.java
      BusmodCreateBusinessClassPropertiesBlobBinaryProperty
      BusmodCreateBusinessClassPropertiesBlobBinaryProperty
      BusmodCreateBusinessClassPropertiesBlobTextPropertyFe
      BusmodCreateBusinessClassPropertiesBlobTextPropertyFe
      BusmodCreateBusinessClassPropertiesBoolPropertyFeatur
      BusmodCreateBusinessClassPropertiesBoolPropertyFeatur
      BusmodCreateBusinessClassPropertiesDateTimePropertyF
      BusmodCreateBusinessClassPropertiesDateTimePropertyF
      BusmodCreateBusinessClassPropertiesDecimalPropertyFe
```

```
/*************************************************************************************
 *
 * Spray diagram definition
 *
 * This file contains the definition of a graphical editor using the Spray Language.
 * Refer to http://code.google.com/a/eclipselabs.org/p/spray/ for documentation.
 *
 * See also busmod.properties to configure generator properties.
 *************************************************************************************/
diagram   busmod

import BusinessDomainDsl.*
import org.eclipse.graphiti.util.IColorConstant
behaviour samen [
    doit "Do it" ;
]

class BusinessClass:
    container  ( fill=RGB(217,228,255) )
    [
          text ( )  { "<<"+eClass.name+">> " + name};
          line ( color=black width=2);
          reference properties attribute dataType;
          line                       (width=1  color=blue);
          reference businessRules;   // will use name property by default
          line                       (width=1  color=blue);

          text () "::" + description;
    ]
    references [
         superclass : connection();
    ]
    behavior [
        create  palette "Shapes" ;
        doWithBusinessClass "Do It With";
        group samen ;
    ]
```

Graphical modelling editors / Java code

Examples

- Textual DSL to specify contracts for model transformations.
- Model transformation to generate further artifacts (e.g. test suites).



```
specification bpmn2pn <BPMN:"BPMN", PN:"petri_nets"> {

   ppattern ParallelGateway1 { // positive invariant
      BPMN {
         t:Task{}
         g:ParallelGateway{}
         s:SequenceFlow{ sourceRef=@t; targetRef=@g; } }
      PN {
         pl:Place{ outarcs=@tr; }
         tr:Transition{}
      }
      condition: "t.name = pl.name";
   }

   npattern ParallelGateway3 { // negative invariant
      BPMN {
        t1:Task{}
        t2:Task{}
        g:ParallelGateway{}
        s1:SequenceFlow{ sourceRef=@t1; targetRef=@g; }
      }
      PN {
         pl1:Place{ outarcs=@tr; }
         pl2:Place{ outarcs=@tr; }
         tr:Transition{}
      }
      condition: "t1.name = pl1.name",
                 "t2.name = pl2.name";
      neg { // disabling condition
         BPMN {
            t2:Task{}
            g:ParallelGateway{}
            s2:SequenceFlow{ sourceRef=@t2; targetRef=@g; }
         }
         PN {}
      }
   }
```

Contracts / EOL

Index

- Introduction.
- **Generation of modelling environments.**
  - **Syntax-directed environments.**
  - **Free-hand environments.**
- Technologies.
- 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.
- More freedom in the way diagrams are edited.
- This can be a disadvantage, as in syntax-directed environments, the allowed editing actions are a guide for users.

Syntax-directed environments

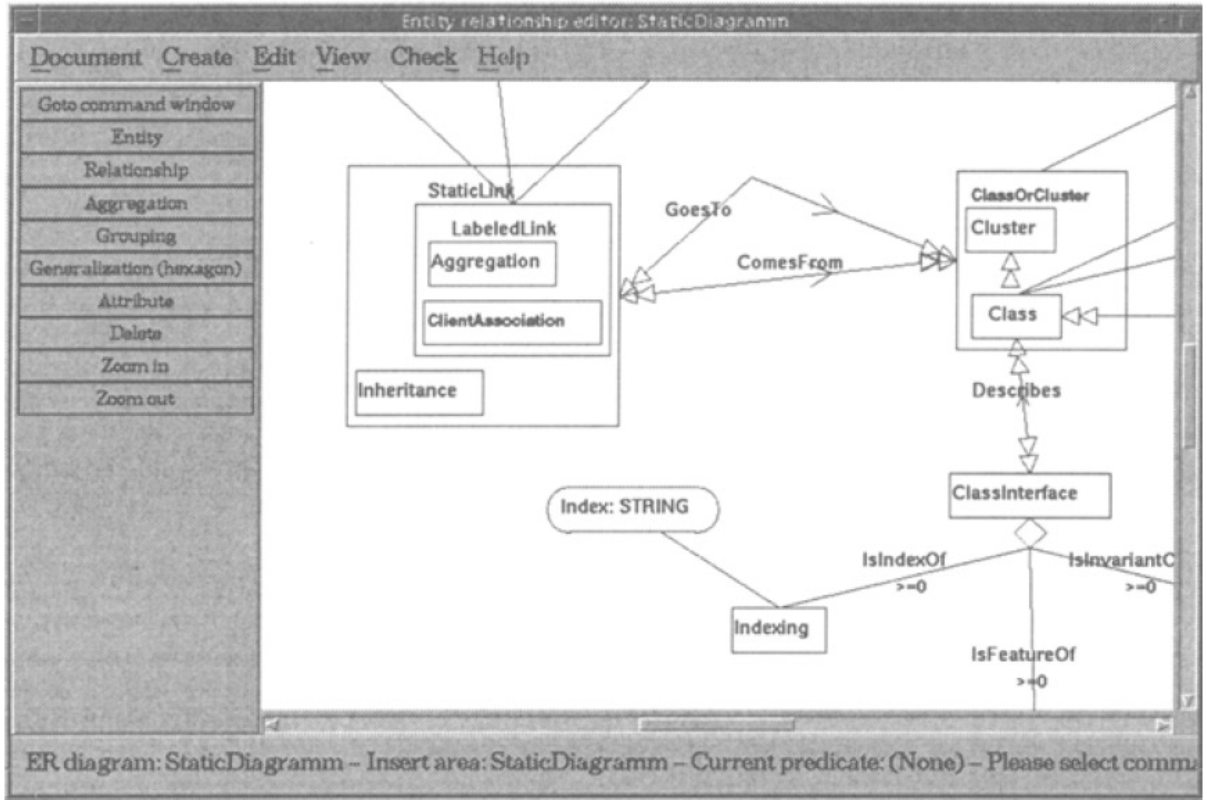
- Editing actions are modelled through graph grammar rules.
- In addition to creation rules, this includes deleting rules.
- Interesting technique if there are complex editing actions (which involve creating or connecting many elements).
- Having many different rules can make this approach difficult to manage.

Index

- Introduction.
- Generation of modelling environments.
- **Technologies.**
- Bibliography.

Tools

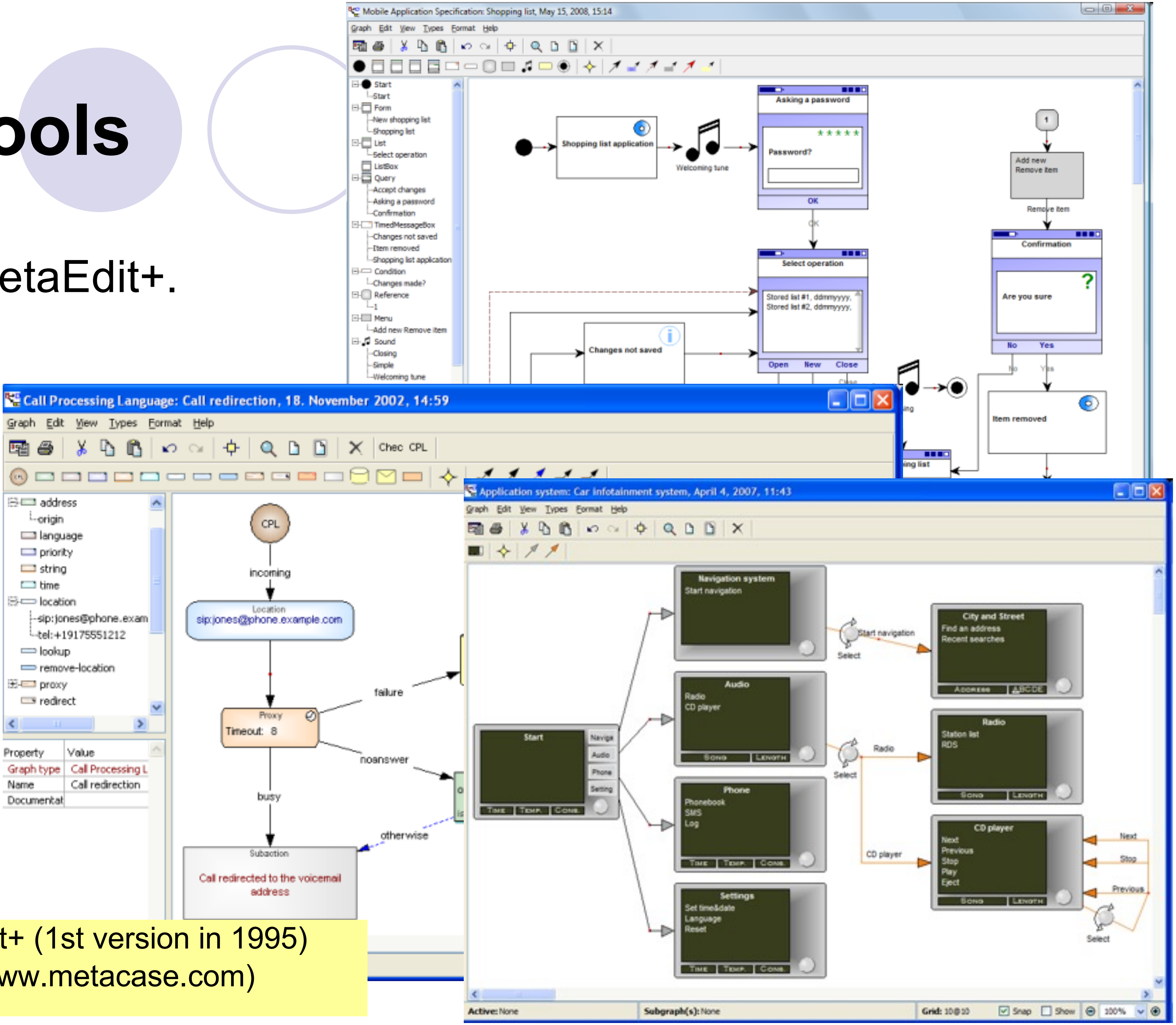
- KOGGE 1997.
- Ebbert, Süttenbach, Uhe (Loblenz).
- Meta-CASE tools, to build CASE tools.



ER diagram: StaticDiagramm – Insert area: StaticDiagramm – Current predicate: (None) – Please select comm

Tools

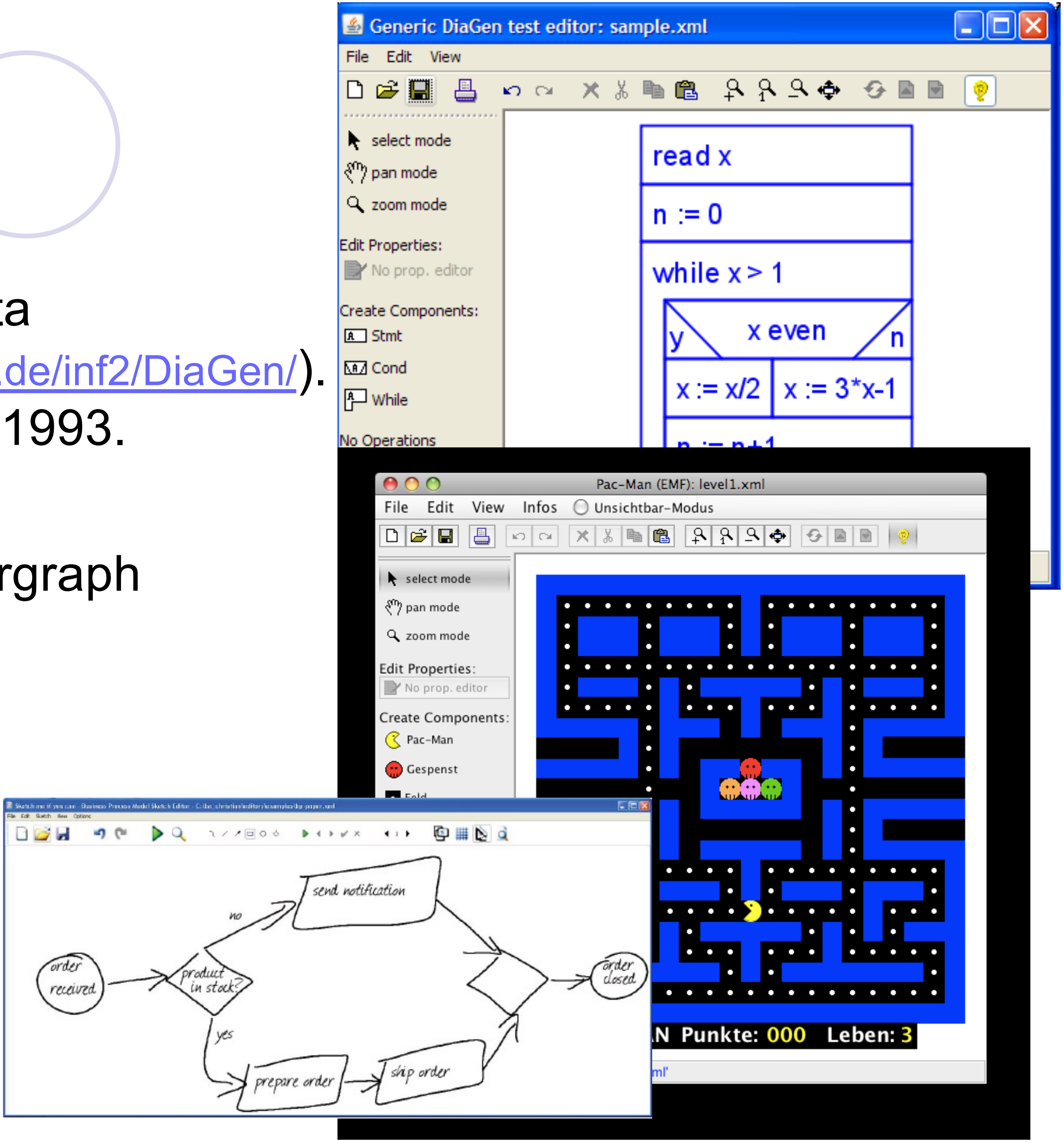
- MetaEdit+.



MetaEdit+ (1st version in 1995)
(http://www.metacase.com)

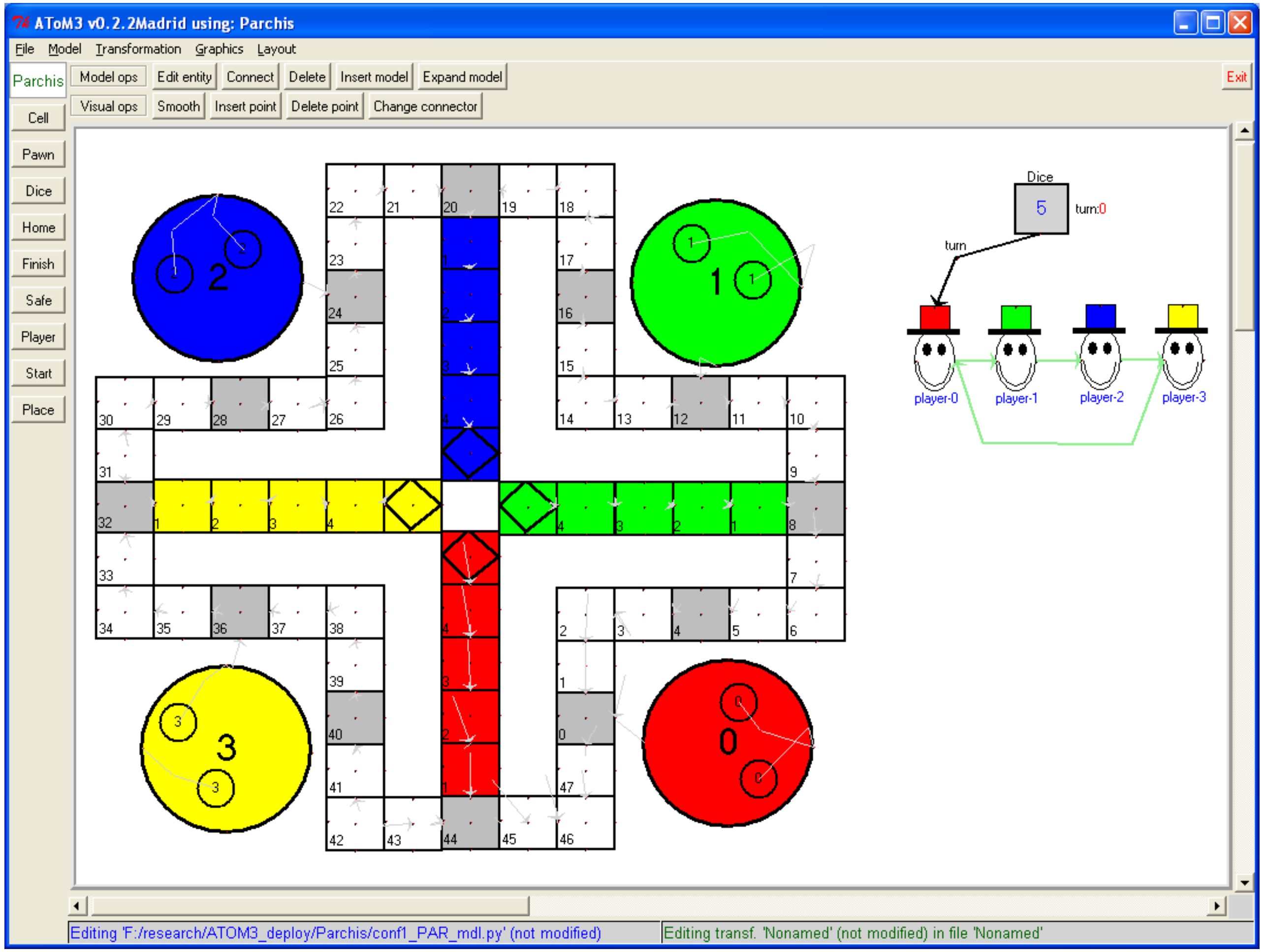
Tools

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



Tools

- AToM^3 (2002)
- Model manipulation can be graphically defined using graph transformation.
- Simulation.

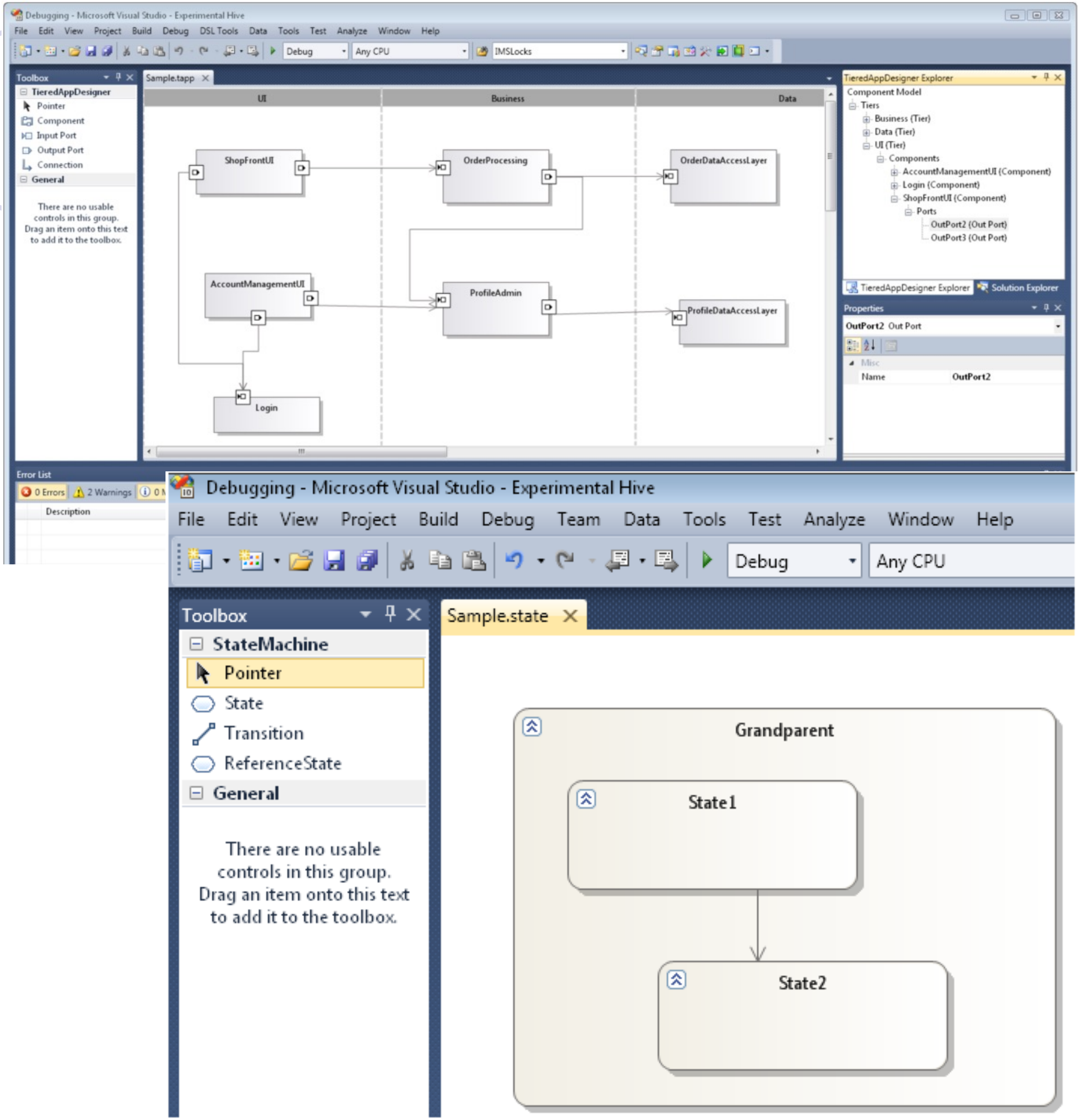


Editing 'F:/research/ATOM3_deploy/Parchis/conf1_PAR_mdl.py' (not modified)

Editing transf. 'Nonamed' (not modified) in file 'Nonamed'

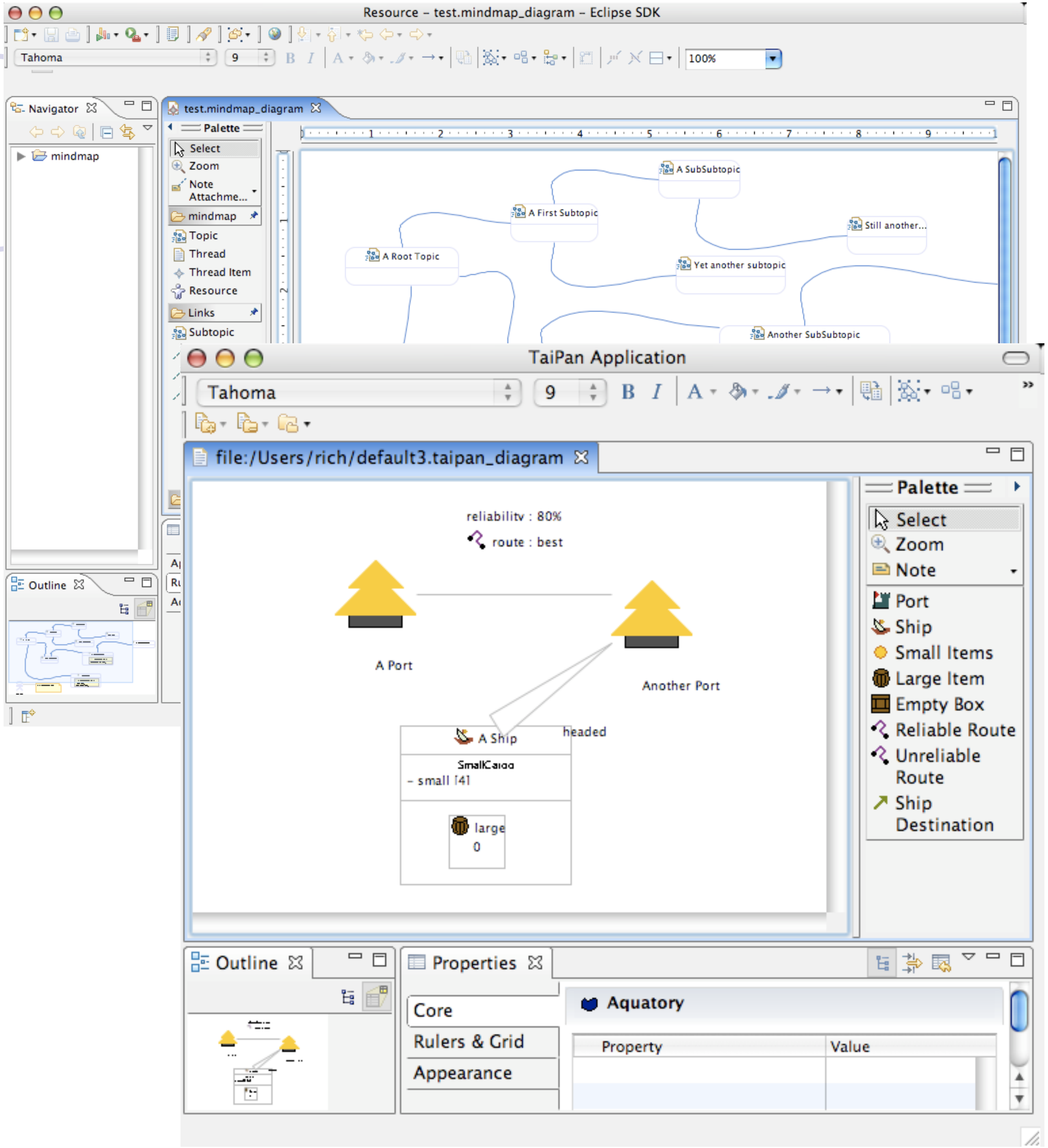
Tools

- DSL Tools.
- Microsoft/ Visual Studio.



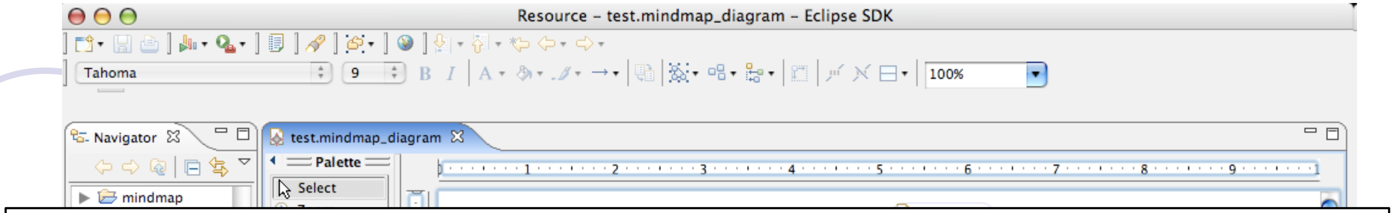
Tools

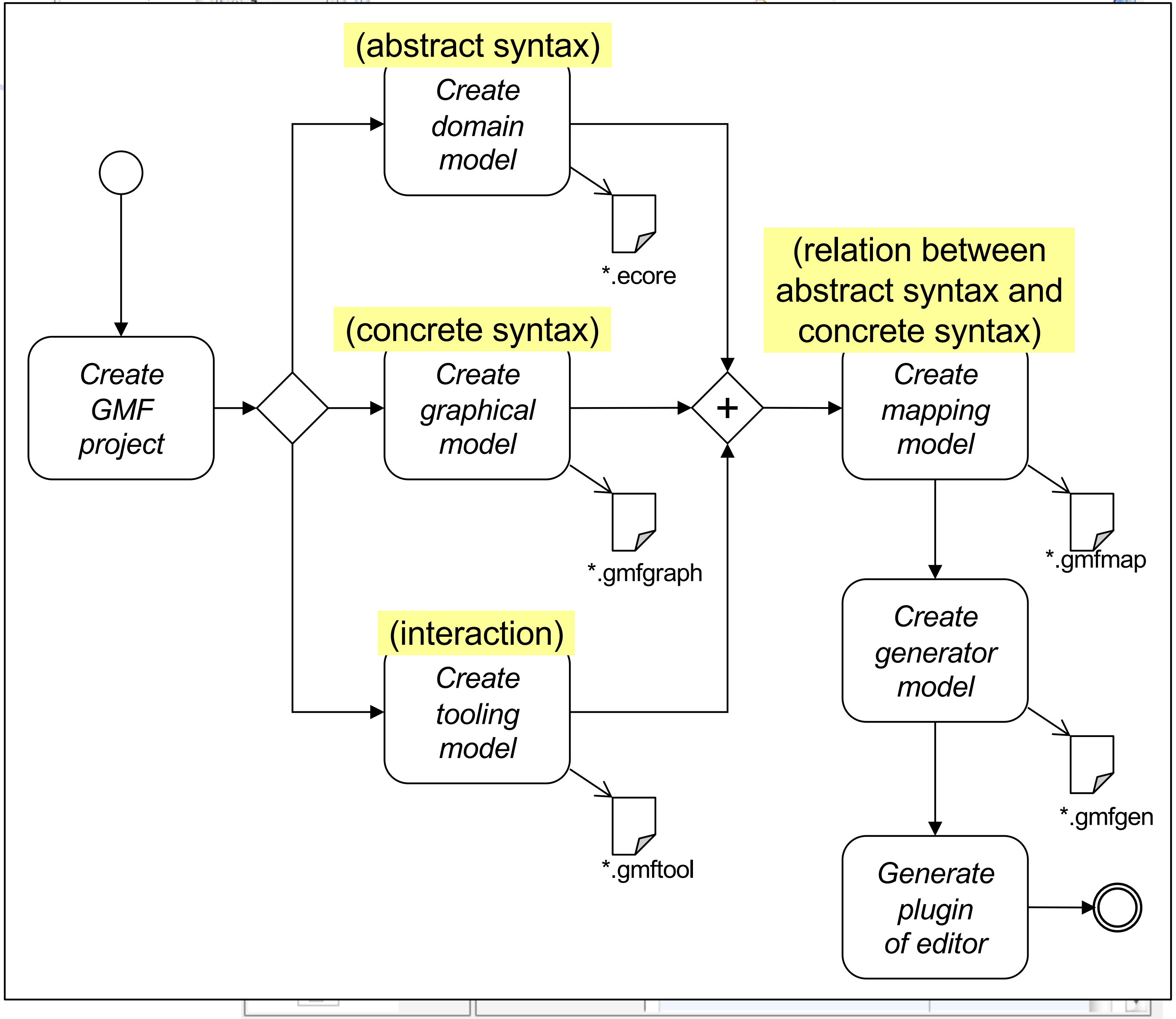
- GMF.
- EMF/Eclipse.
- More complex!



Tools

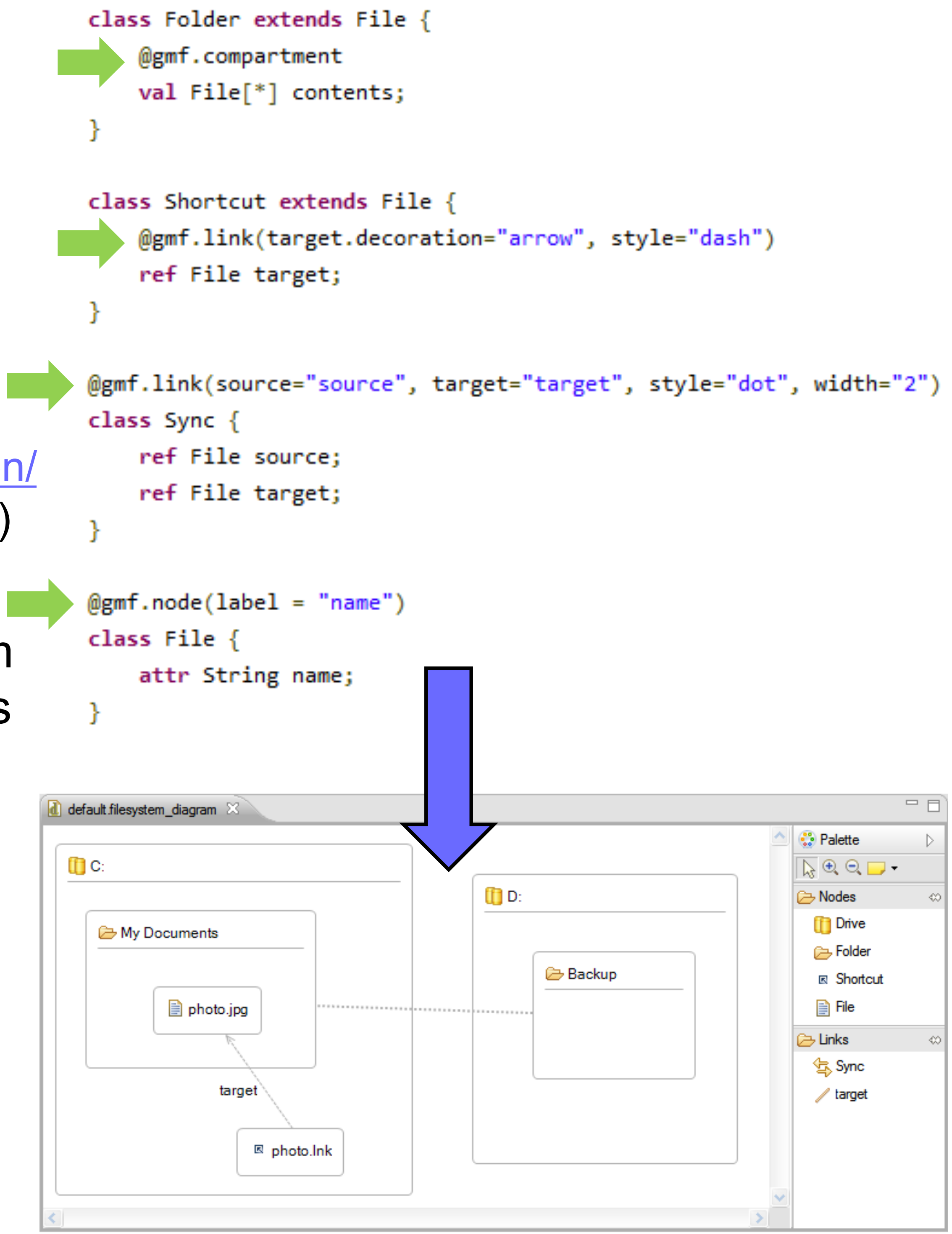
- GMF.
- EMF/Eclipse.
- More complex!





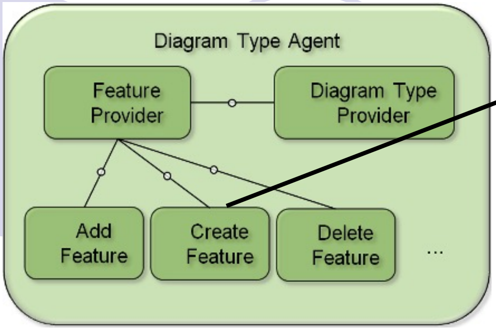
Tools

- Eugenia (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



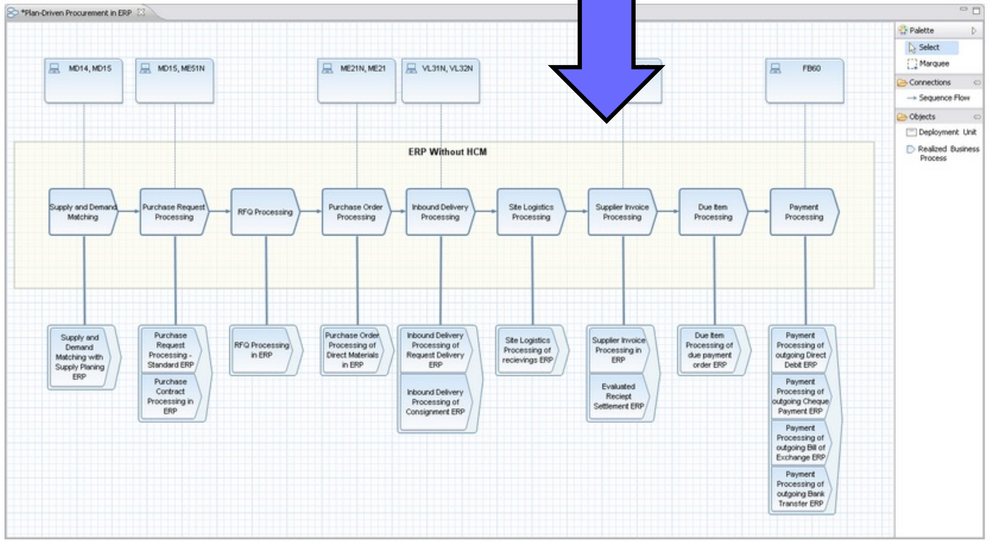
```
class Folder extends File {
    @gmf.compartment
    val File[*] contents;
}

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;
}
```

Tool



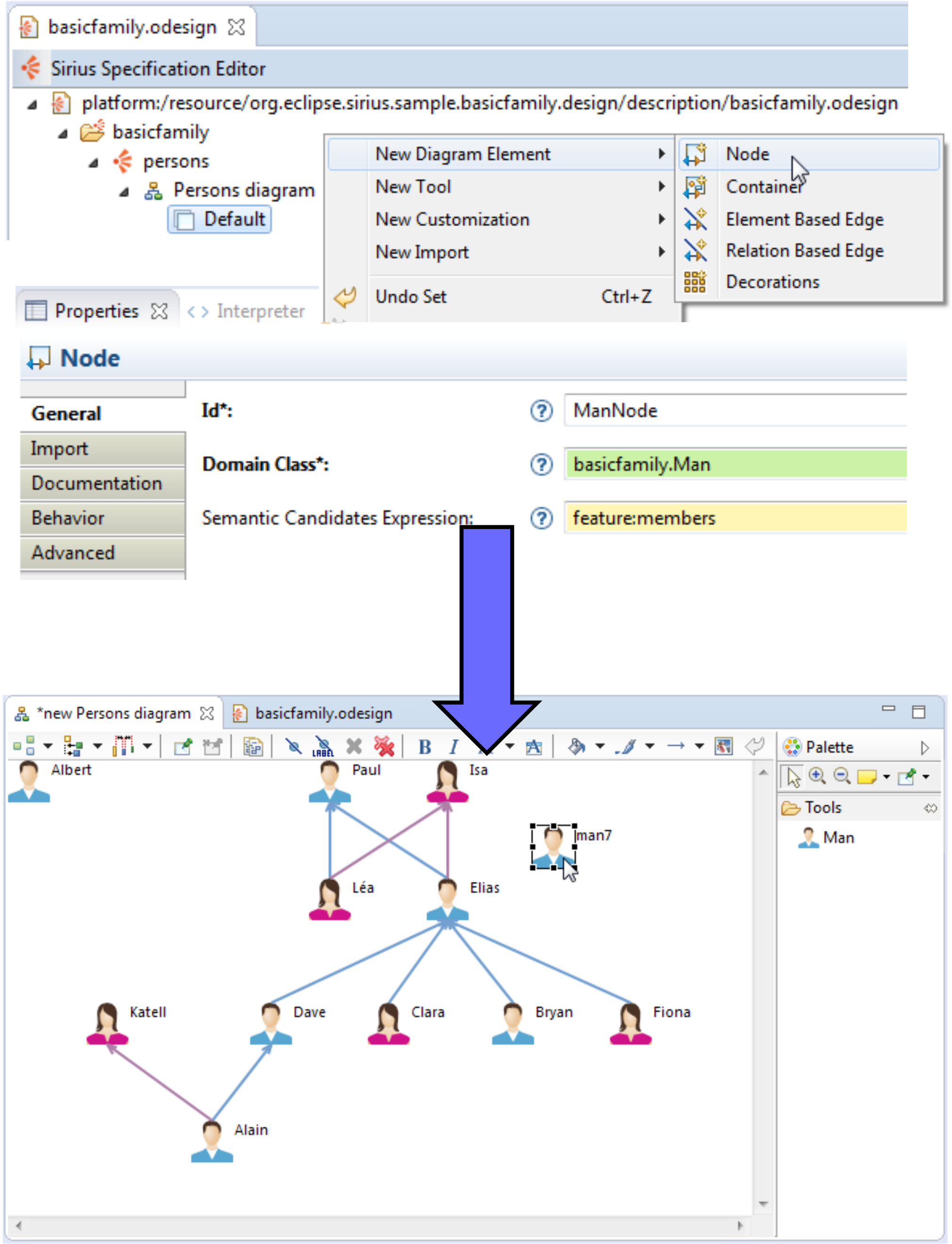
- Graphiti (http://www.eclipse.org/graphiti/)
- Flat learning curve (Java API + Graphiti objects), high flexibility, common look and feel with sensible defaults
- Spray (https://code.google.com/a/eclipselabs.org/p/spray/): DSL to describe Graphiti editors

```
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, ne     chaseOrder);
}
```


*Plan-Driven Procurement in ERP
Palette
Select
Marquee
Connections
Sequence Flow
Objects
Deployment Unit
Realized Business Process
MD14, MD15
MD15, ME51N
ME21N, ME21
VL31N, VL32N
FB60
ERP Without HCM
Supply and Demand Matching
Purchase Request Processing
RFQ Processing
Purchase Order Processing
Inbound Delivery Processing
Site Logistics Processing
Supplier Invoice Processing
Due Item Processing
Payment Processing
Supply and Demand Matching with Supply Planing ERP
Purchase Request Processing - Standard ERP
Purchase Contract Processing in ERP
RFQ Processing in ERP
Purchase Order Processing of Direct Materials in ERP
Inbound Delivery Processing of Request Delivery ERP
Inbound Delivery Processing of Consignment ERP
Site Logistics Processing of recievings ERP
Supplier Invoice Processing in ERP
Evaluated Reciept Settlement ERP
Due Item Processing of due payment order ERP
Payment Processing of outgoing Direct Debit ERP
Payment Processing of outgoing Cheque Payment ERP
Payment Processing of outgoing Bill of Exchange ERP
Payment Processing of outgoing Bank Transfer ERP

Tools

- Sirius (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.



basicfamily.odesign

Sirius Specification Editor

platform:/resource/org.eclipse.sirius.sample.basicfamily.design/description/basicfamily.odesign
- basicfamily
  - persons
    - Persons diagram
      - Default

Node

Id*: ManNode

Domain Class*: basicfamily.Man

Semantic Candidates Expression: feature:members

Index

- Introduction.
- Generation of modelling environments.
- Technologies.
- **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".