4. How to Transform Models with Graph Rewriting

1) Graph Transformations
2) Programmed Graph Rewriting

Prof. Dr. U. Aßmann
Technische Universität Dresden
Institut für Software- und Multimediatechnik
Gruppe Softwaretechnologie
http://st.inf.tu-dresden.de

Version 15-1.1, 02.01.16

Part of ST2 and MOST

Obligatory Reading

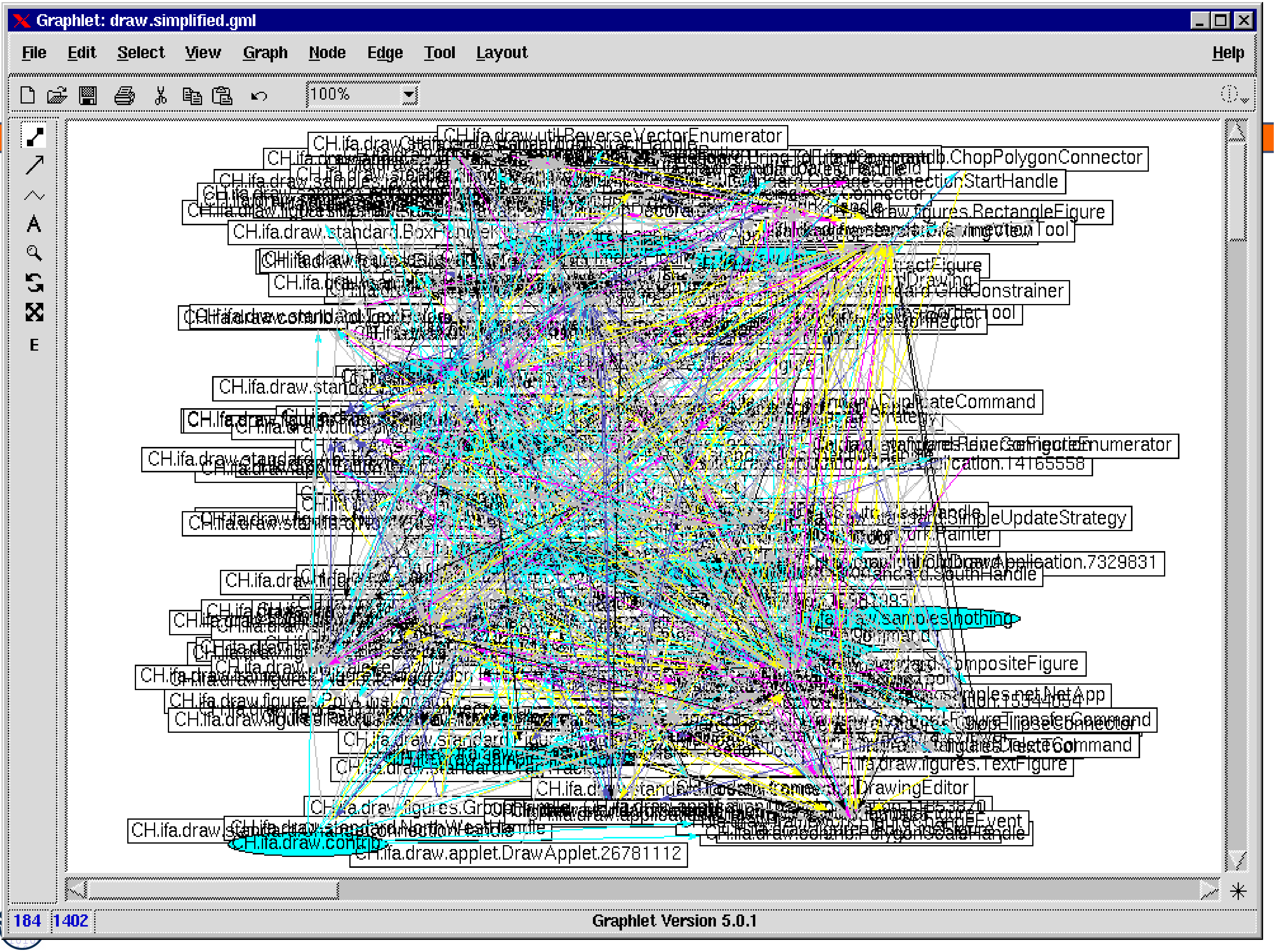
- Jazayeri Chap 3. If you have other books, read the lecture slides carefully and do the exercise sheets
- T. Mens. On the Use of Graph Transformations for Model Refactorings. In GTTSE 2005, Springer, LNCS 4143
  - http://www.springerlink.com/content/5742246115107431/
- F. Klar, A. Königs, A. Schürr: "Model Transformation in the Large", Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on the foundations of software engineering, New York: ACM Press, 2007; ACM Digital Library Proceedings, 285-294. http://www.idt.mdh.se/esec-fse-2007/
- www.fujaba.de www.moflon.org
- T. Fischer, J. Niere, L. Torunski, and A. Zündorf, 'Story Diagrams: A new Graph Rewrite Language based on the Unified Modeling Language', in Proc. of the 6th International Workshop on Theory and Application of Graph Transformation (TAGT), Paderborn, Germany (G. Engels and G. Rozenberg, eds.), LNCS 1764, pp. 296--309, Springer Verlag, November 1998. http://www.upb.de/cs/ag-schaefer/Veroeffentlichungen/Quellen/Papers/1998/TAGT1998.pdf

Obligatory Literature

- View models (Wikipedia)
  - http://en.wikipedia.org/wiki/View_model
- [Kruchten] Kruchten, P., Vancouver, B., C.: The 4+1 View Model of Architecture; IEEE Software, 12 (6), Nov. 1995, IEEE, S. 42-50
  - http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=469759

Further Reading

- Reducible graphs
  - [ASU86] Alfred A. Aho, R. Sethi, and Jeffrey D. Ullman. Compilers: Principles, Techniques, and Tools. Addison-Wesley, 1986.
- Search for these keywords at
  - http://scholar.google.com
  - http://citeseer.ist.psu.edu
  - http://portal.acm.org/guide.cfm
  - http://ieeexplore.ieee.org/
  - http://www.gi-ev.de/wissenschaft/digitbibl/index.html
  - http://www.springer.com/computer?SGWID=1-146-0-0-0


Graphlet: draw.simplified.gml
File
Edit
Select
View
Graph
Node
Edge
Tool
Layout
Help
100%
CH.ifa.draw.util.ReverseVectorEnumerator
CH.ifa.draw.contrib.ChopPolygonConnector
CH.ifa.draw.standard.ChangeConnectionStartHandle
CH.ifa.draw.figures.RectangleFigure
CH.ifa.draw.standard.BoxHandleKit
CH.ifa.draw.standard.GridConstrainer
CH.ifa.draw.standard.DuplicateCommand
CH.ifa.draw.standard.SimpleUpdateStrategy
CH.ifa.draw.standard.SouthHandle
CH.ifa.draw.samples.nothing
CH.ifa.draw.standard.CompositeFigure
CH.ifa.draw.samples.net.NetApp
CH.ifa.draw.figures.TextFigure
CH.ifa.draw.standard.NorthWestHandle
CH.ifa.draw.contrib
CH.ifa.draw.applet.DrawApplet.26781112
CH.ifa.draw.figures.GroupFigure
CH.ifa.draw.figures.PolyLineConnector
184
1402
Graphlet Version 5.0.1

The Problem: How to Master Large Models in Requirements Engineering, Design, Domain Modeling

- Large models have large graphs
- They can be hard to understand -> models must be simplified
  - By transformation
  - By refactoring (behavior-preserving transformation)
  - By structurings
- To this end, we apply **graph analysis and rewriting**
- Figures taken from Goose Reengineering Tool, analysing a Java class system [Goose, FZI Karlsruhe]

Typical Problems in Software Engineering

- Question: How to Treat the Models of a big Swiss Bank?
  - 25 Mio LOC
  - 170 terabyte databases
- Question: How to Treat the Models of a big Operating System?
  - 25 Mio LOC
  - thousands of variants
- Requirements for Modelling in Requirements and Design
  - We need automatic structuring methods
  - We need help in restructuring by hand...
- Motivations for structuring
  - Getting better overview
  - Comprehensibility
  - Validatability, Verifiability

??

4.1 Graph Transformations

Model and Code Transformations in General

- Model and code transformations, such as refactorings, lowerings, higherings, optimizers, and other transformations can be specified by graph transformations [Mens]

	Horizontal	Vertical
Endogeneous (within one language)	Structurings, Refactorings	Syntactic and semantic refinement
Exogeneous (crossing languages)	Language migration	Generation of platform-specific models (PSM) (see chapter MDA)
		Generation of platform-specific implementation models (PSI)
		Generation of platform-specific implementation (code generation)

Idea: Structure the Software Systems With Graph Rewrite Systems

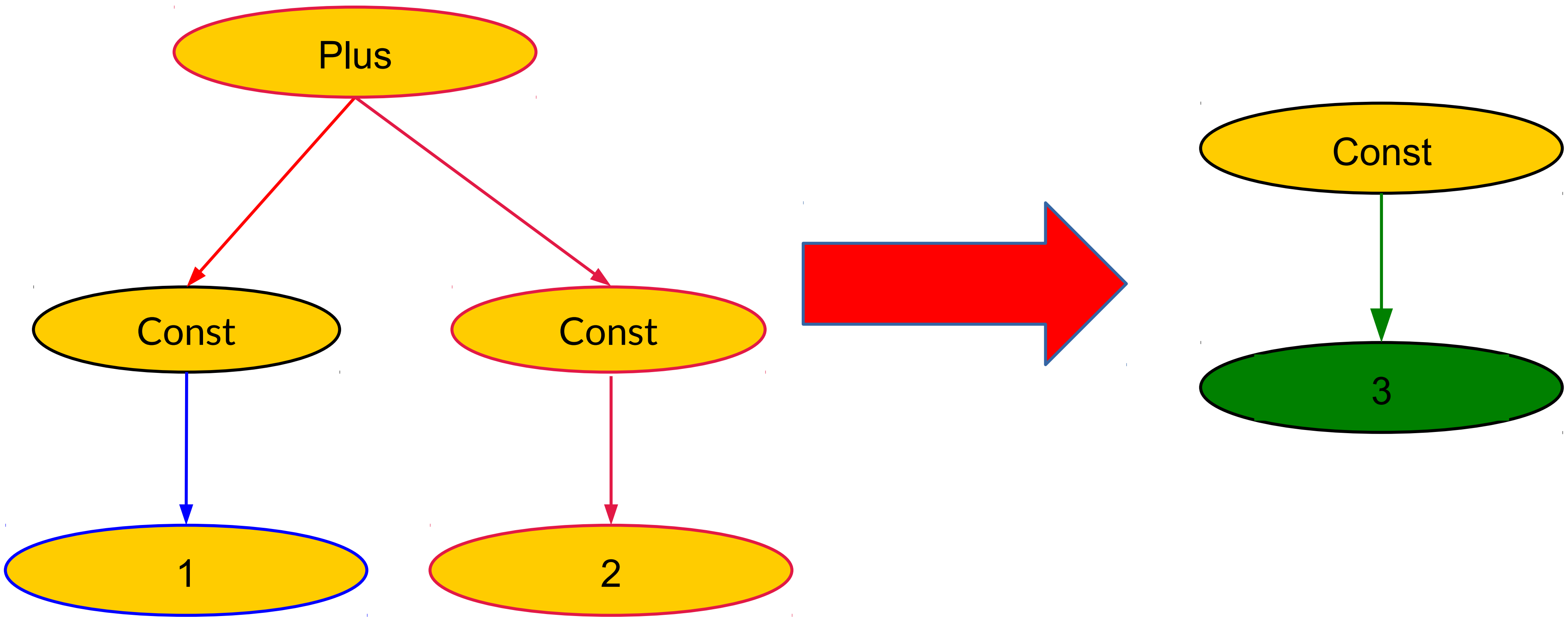
- Graph transformations can be specified by graph rewrite systems (GRS)
  - Or by a programming language, of course
- If a graph transformation only manipulate edges, it can be described with Edge Addition Rewrite Systems (EARS)
- Otherwise, we arrive at general GRS
  - Transformation of complex structures to simple ones
  - Structure complex models and systems

Graph Rewrite Systems

- A **graph rewrite system** G = (S) consists of
  - A **set of rewrite rules S**
    - A rule r = (L,R) consists of 2 graphs L and R (left and right hand side)
    - Nodes of left and right hand side must be identified to each other
    - L = “Mustergraphen” ; R = Ersetzungsgraph”
  - An **application algorithm A**, that applies a rule to the manipulated graph
    - There are many of those application algorithms…

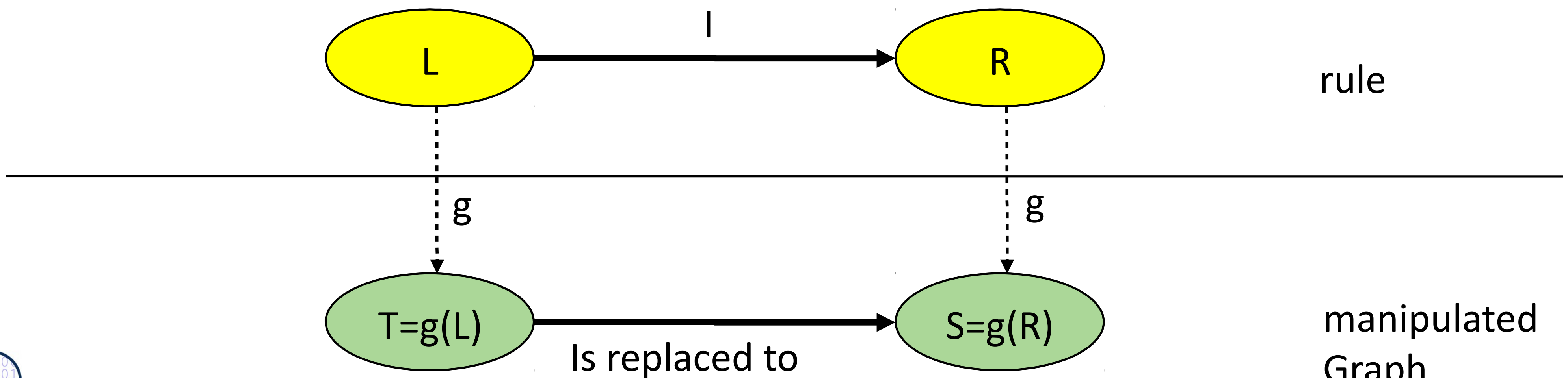
- A **graph rewrite problem** P = (G,Z) consists of
  - A graph rewrite system G
  - A start graph Z
  - One or several result graphs
  - A derivation under P consists of a sequence of applications of rules (direct derivations)
- GRS offer automatic graph rewriting
  - A GRS applies a set of Graph rewrite rules until nothing changes anymore (to the fixpoint, chaotic iteration)
  - Problem: Termination and Uniqueness of solution not guaranteed

Constant Folding as Graph Rewrite Rule

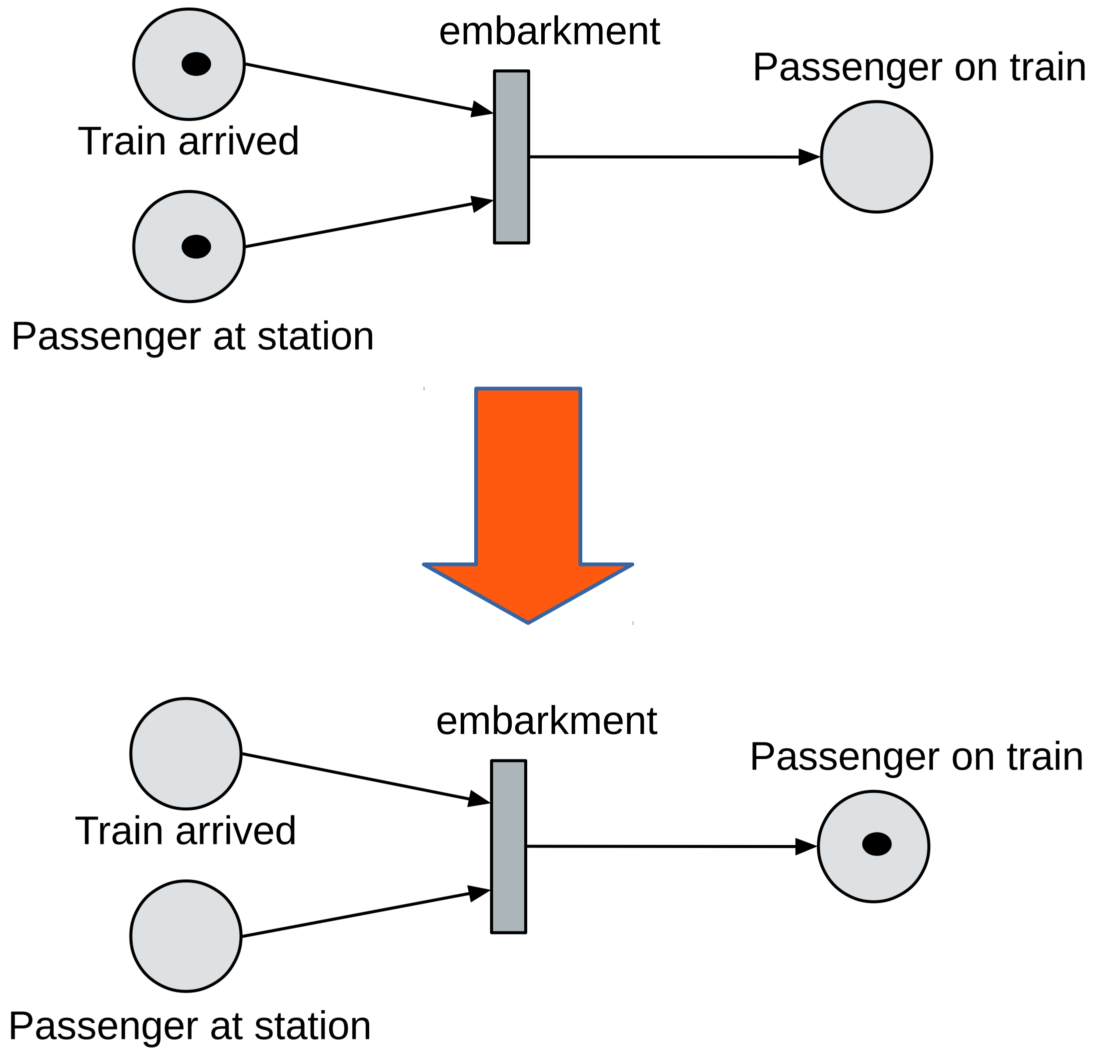


Application of a Graph Rewrite Rule

- **Match** the left hand side: Look for a subgraph T of the manipulated graph: look for a graph morphism g with g(L) = T
- Evaluate **side conditions** of the left hand side
- Evaluate **right hand side**
  - Delete all nodes and edges that are no longer mentioned in R
  - Allocate new nodes and edges from R, that do not occur in L
- **Embedding**: redirect certain edges from L to new nodes in R
  - Resulting in S, the mapping of g(R)
- Evaluate **side actions**
  - Assign attributes to nodes

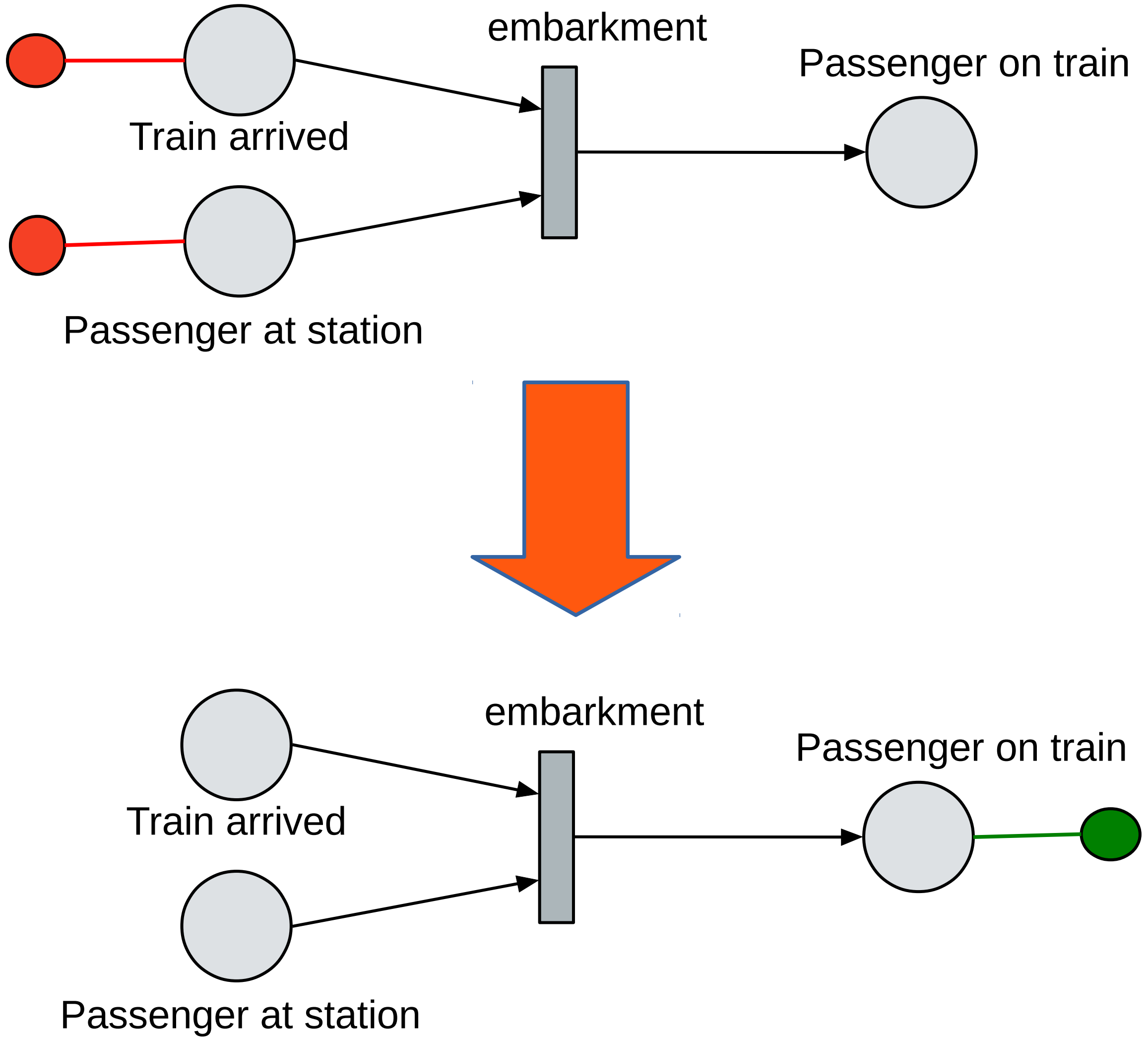


The Firing Rule of Petri Nets is a Graph Rewriting Rule



The Firing Rule of Petri Nets is a Graph Rewriting Rule

- Tokens can be modeled as special nodes attached to places
- The application of the rewrite rule models an event



Different Kinds of Graph Transformation Systems

- **Automatic** Graph Rewriting
  - Iteration of rules until termination
- Graph **Reduction**: Reducing a graph; rewrite system only has reductive rules
- **Programmed** Graph Rewriting: The rules are applied of a control flow program. This program guarantees termination and selects one of several solutions
  - Examples: PROGRES from Aachen/München
  - Fujaba on UML class graphs, from Paderborn, Kassel www.fujaba.de
  - MOFLON from Darmstadt www.moflon.org
- **Strategic** Graph Rewriting:
  - The rules are applied by strategies, higher-order functions and recursion strategies, such as bottom-up / top-down
- Graph **grammars** (Graph **Recognition**)
  - Special variant of automatic graph rewrite systems
  - Graph grammars contain in their rules and in their generated graphs special nodes, so called non-terminals
  - A result graph must not have non-terminals
  - In analogue to String grammars, derivations can be formed and derivation trees

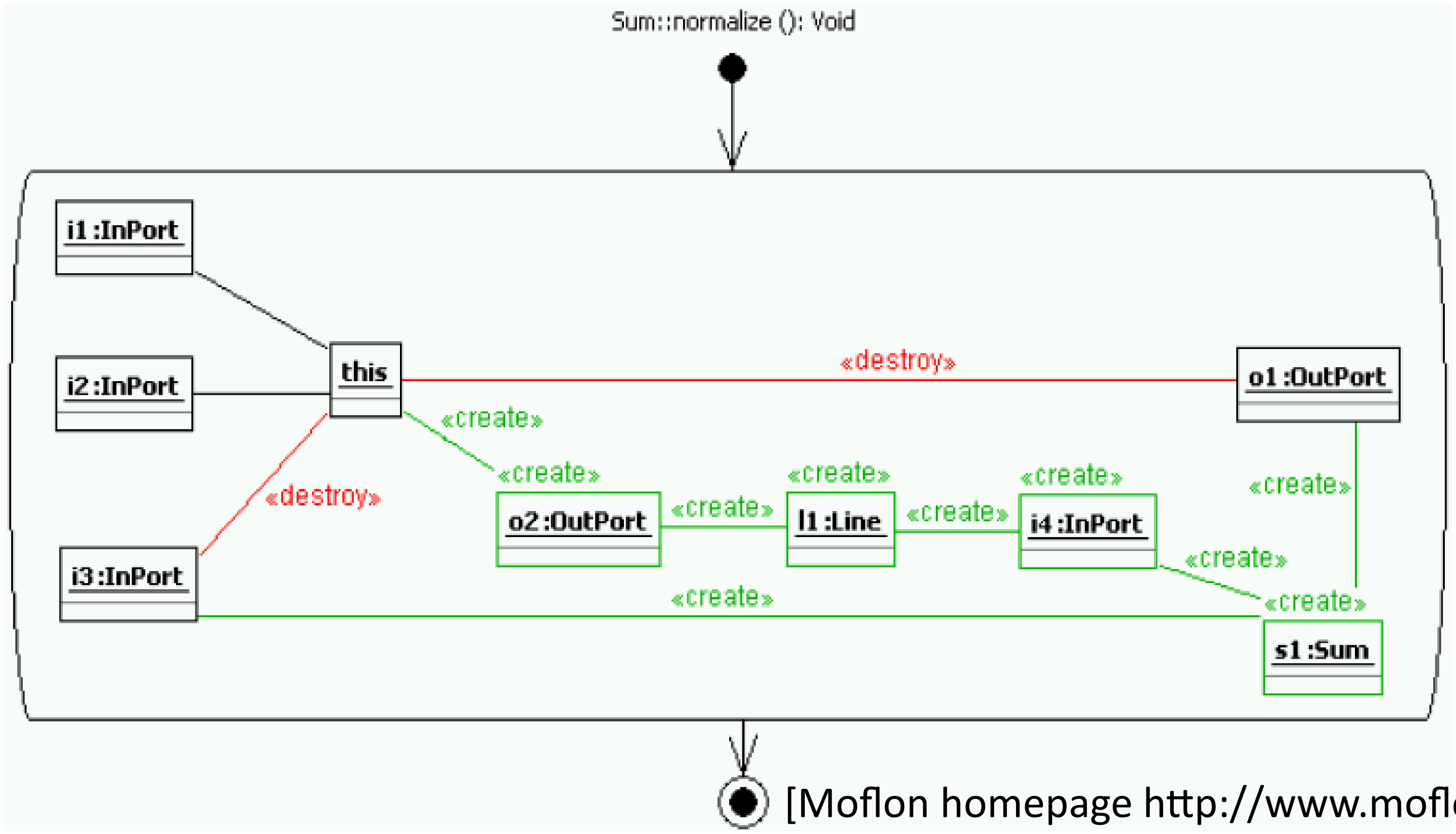
Simpler Forms of Transformation Systems: TRS and Context-Sensitive TRS

- **Term rewriting** replaces terms (ordered trees)
  - right and left hand sides are unordered trees or (ordered) terms
  - Application: XML terms
- **Ground** term rewrite systems, GTRS: only ground terms in left hand sides
  - A GTRS always works bottum-up on the leaves of a tree
  - For GTRS there are very fast, linear algorithms
- **Variable** term rewrite systeme, VTRS: terms with variables
  - Replacement everywhere in the tree
- **Dag** rewrite systems (DAGRS)
  - If a term contains a variable twice (non-linear), it specifies a dag
  - Dag rewrite systems containt dags in left and right hand sides (non-linear term rewriting)
- **Context-sensitive** Term Rewriting
  - e.g., remote-attribute controlled-rewriting (RACR)
  - Analyse the context and rewrite trees based on context attributes

4.2 Programmed Graph Rewriting

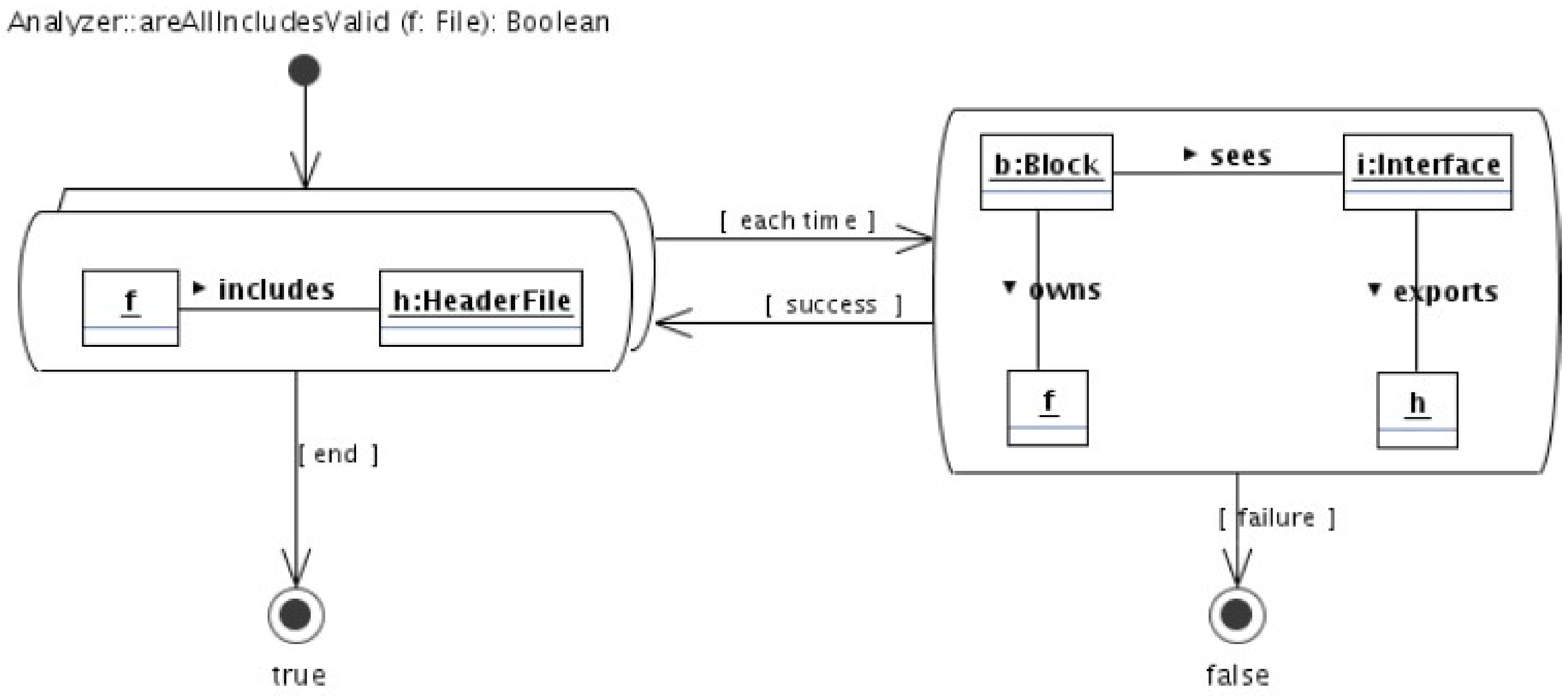
MOFLON and Fujaba

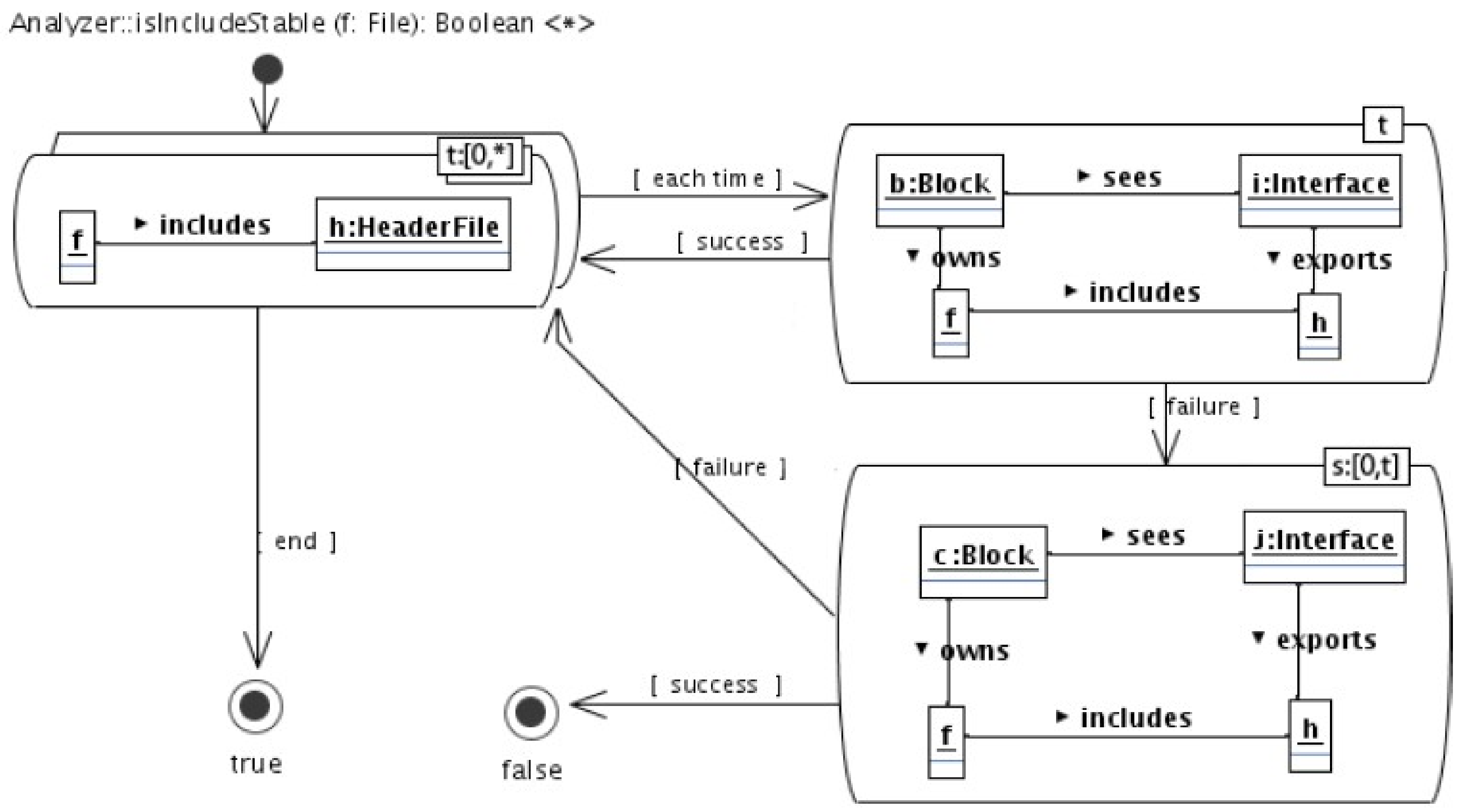
- MOFLON and Fujaba embed graph rewrite rules into activity diagrams (aka storyboards)
  - A rule set executes as an atomic activity
  - Colors express actions



[Moflon homepage http://www.moflon.org]
[Fujaba homepage http://www.fujaba.de]

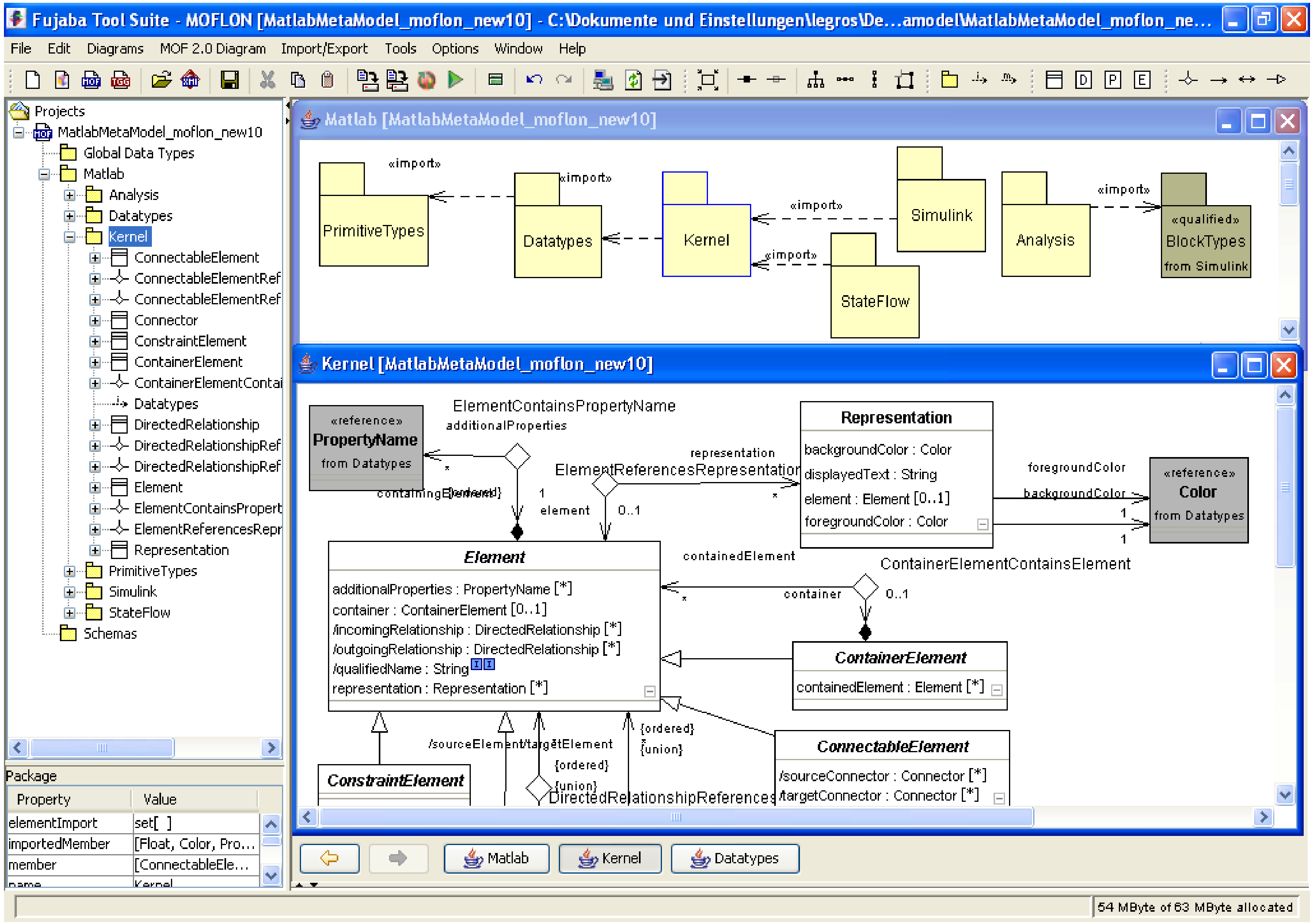
Storyboards are Procedures (Activity Diagrams) with Graph Rewrite Rules In Activities




Analyzer::isIncludeStable (f: File): Boolean <*>
t:[0,*]
f
includes
h:HeaderFile
[each time]
[success]
t
b:Block
sees
i:Interface
owns
exports
includes
f
h
[failure]
s:[0,t]
c:Block
sees
j:Interface
owns
exports
includes
f
h
[failure]
[end]
[success]
true
false

MOFLON

- Works on graphs typed by metamodels, specified in MOF


Fujaba Tool Suite - MOFLON [MatlabMetaModel_moflon_new10] - C:\Dokumente und Einstellungen\legros\De...amodel\MatlabMetaModel_moflon_ne...
File Edit Diagrams MOF 2.0 Diagram Import/Export Tools Options Window Help
Projects
MatlabMetaModel_moflon_new10
Global Data Types
Matlab
Analysis
Datatypes
Kernel
ConnectableElement
ConnectableElementRef
ConnectableElementRef
Connector
ConstraintElement
ContainerElement
ContainerElementContai
Datatypes
DirectedRelationship
DirectedRelationshipRef
DirectedRelationshipRef
Element
ElementContainsPropert
ElementReferencesRepr
Representation
PrimitiveTypes
Simulink
StateFlow
Schemas
Matlab [MatlabMetaModel_moflon_new10]
PrimitiveTypes
«import»
Datatypes
«import»
Kernel
«import»
Simulink
«import»
StateFlow
Analysis
«import»
«qualified» BlockTypes from Simulink
Kernel [MatlabMetaModel_moflon_new10]
«reference» PropertyName from Datatypes
ElementContainsPropertyName
additionalProperties
containingElement
{ordered}
1
element
0..1
representation
ElementReferencesRepresentation
Representation
backgroundColor : Color
displayedText : String
element : Element [0..1]
foregroundColor : Color
foregroundColor
backgroundColor
1
1
«reference» Color from Datatypes
Element
additionalProperties : PropertyName [*]
container : ContainerElement [0..1]
/incomingRelationship : DirectedRelationship [*]
/outgoingRelationship : DirectedRelationship [*]
/qualifiedName : String
representation : Representation [*]
containedElement
ContainerElementContainsElement
container
0..1
ContainerElement
containedElement : Element [*]
/sourceElement
/targetElement
{ordered}
{union}
{ordered}
{union}
ConstraintElement
DirectedRelationshipReferences
ConnectableElement
/sourceConnector : Connector [*]
/targetConnector : Connector [*]
Matlab
Kernel
Datatypes
54 MByte of 63 MByte allocated

Package

Property	Value
elementImport	set[ ]
importedMember	[Float, Color, Pro...
member	[ConnectableEle...
name	Kernel

Other Software Engineering Applications

- Modeling state based systems (such as Petrinets)
- Mapping a PIM to a PSM in Model-Driven Architecture (transformational design)
- Graph Structurings (for simplification of models)
- Refactorings (see also Course DPF)
- Semantic refinements (transformational design)
- Round-Trip Engineering (RTE)

The End: What Have We Learned

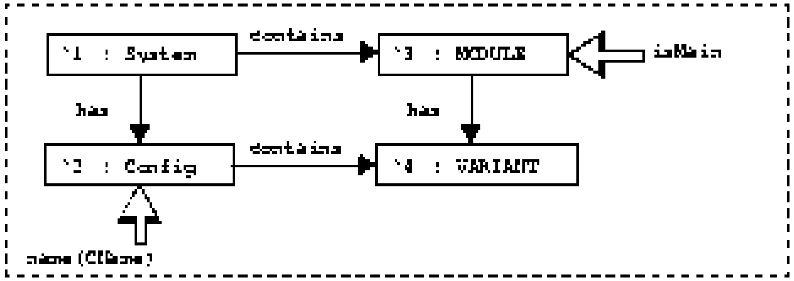
- Graph rewrite systems are tools to transform graph-based models and graph-based program representations
- Petrinets are simple graph rewrite systems
- Graph rewriting can be automatic, programmed, strategic, reductive, recognizing

04.A.1
PROGRES, the GRS tool from the IPSEN Project

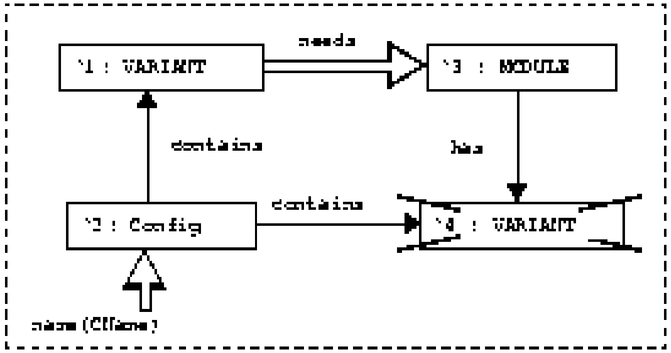
- PROGRES has been the firstl tool to model graph algorithms by graph rewriting
- Textual and graphical editing
- Code generation in several languages

- http://www-i3.informatik.rwth-aachen.de/tikiwiki/tiki-index.php?page_ref_id=213

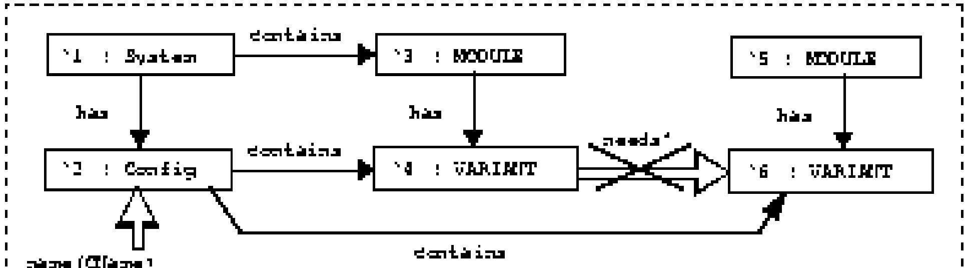
```
query ConsistentConfiguration( out CName : string ) =

    (* A configuration is consistent if:
    (* 1) it contains a variant of the system's main module,  *)
    (* 2) it contains a variant for any module which is       *)
    (*     needed by another included variant, and            *)
    (* 3) it does not contain variants which are not needed   *)
    (*     by needed variants.                                *)

   use LocalName: string do
      ConfigurationWithMain( out LocalName )
    & not UnresolvedImportExists( LocalName )
    & not ConfigurationWithUselessVariant( LocalName )
    & CName := LocalName
   end

end;

test ConfigurationWithMain( out CName : string )  =
```

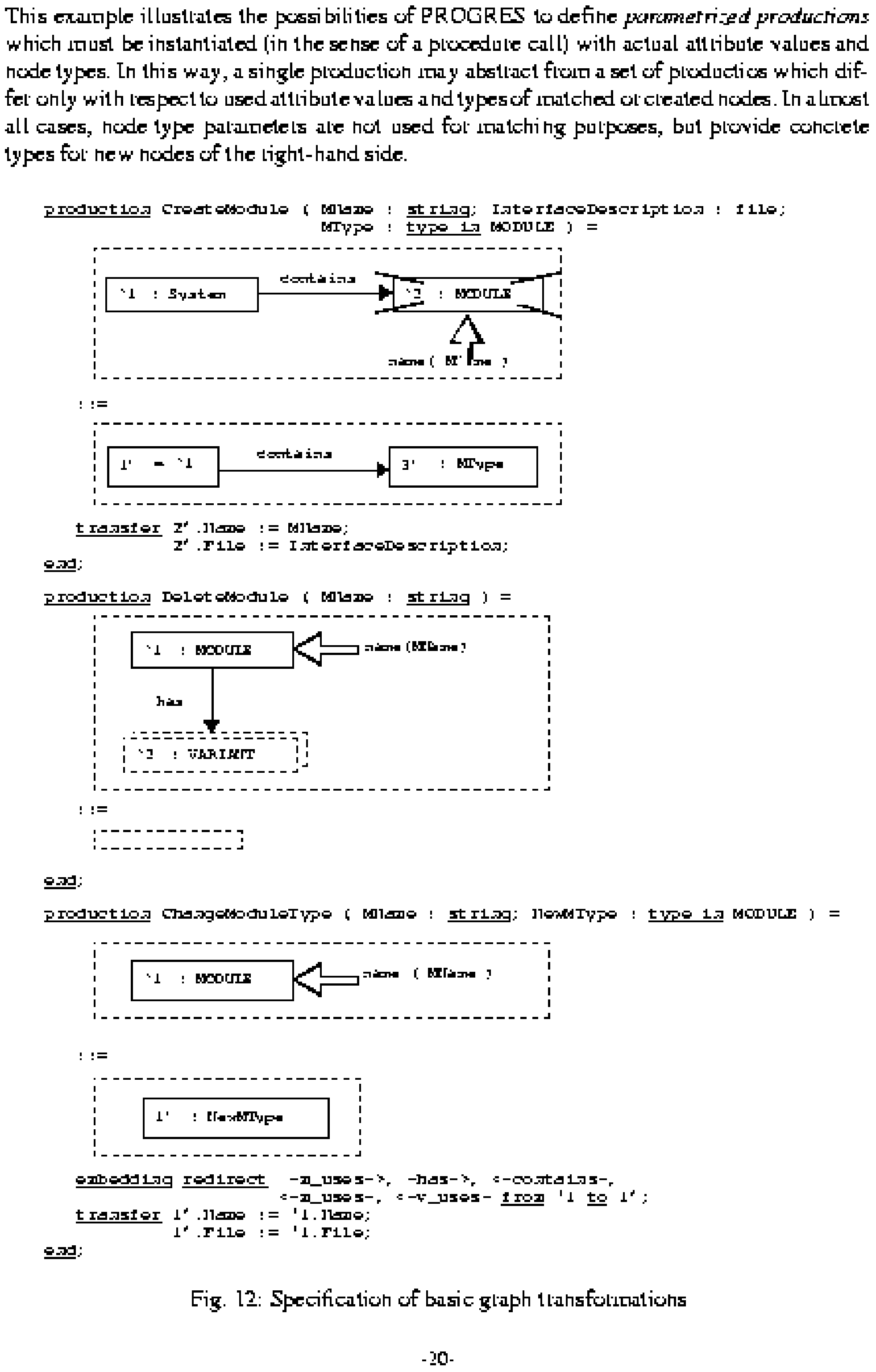


```
end;

test UnresolvedImportExists( CName : string ) =
```



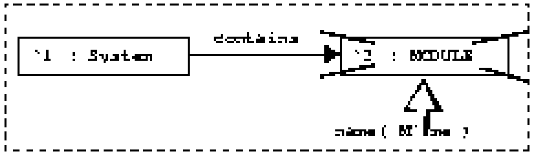
```
end;

test ConfigurationWithUselessVariant( CName : string )  =
```



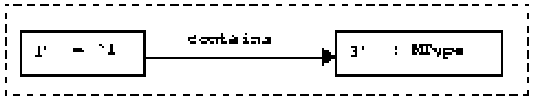


This example illustrates the possibilities of PROGRES to define *parametrized productions* which must be instantiated (in the sense of a procedure call) with actual attribute values and node types. In this way, a single production may abstract from a set of productions which differ only with respect to used attribute values and types of matched or created nodes. In almost all cases, node type parameters are not used for matching purposes, but provide concrete types for new nodes of the right-hand side.

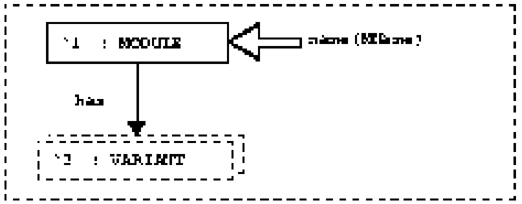
```
production CreateModule ( MName : string; InterfaceDescription : file;
                          MType : type in MODULE ) =
```



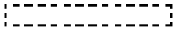
```
   ::=
```



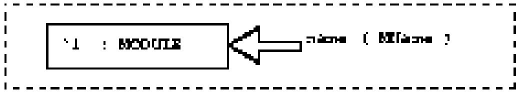
```
   transfer 2'.Name := MName;
            2'.File := InterfaceDescription;
end;

production DeleteModule ( MName : string ) =
```



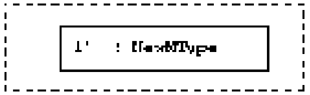
```
   ::=
```



```
end;

production ChangeModuleType ( MName : string; NewMType : type in MODULE ) =
```



```
   ::=
```

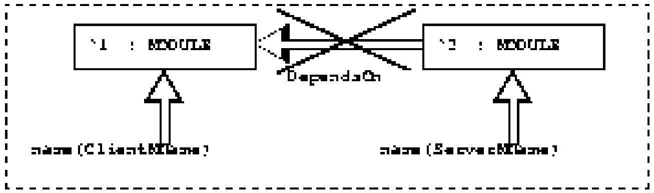


```
   embedding redirect  -m_uses->, -has->, <-contains-,
                       <-m_uses-, <-v_uses- from `1 to 1';
   transfer 1'.Name := `1.Name;
            1'.File := `1.File;
end;
```

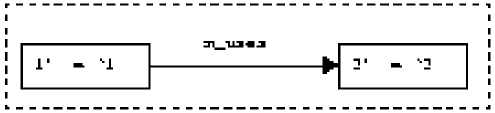
Fig. 12: Specification of basic graph transformations

-20-

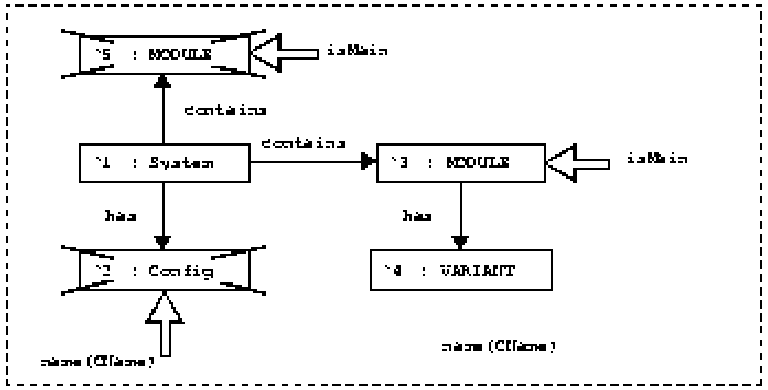
```
production CreateMUse( ClientMName, ServerMName : string ) =
```



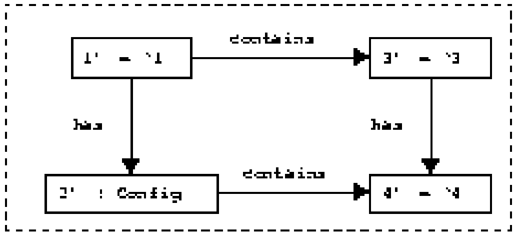
```
  ::=
```



```
end;

production InitConfig(     CName : string;
                        ReqProps : string [0:n];
                    out NewProps : string [0:n] ) =
```



```
  ::=
```



```
   condition '4.Props 'are_in' ReqProps;
   transfer   2'.Name := CName;
              2'.Props := ReqProps;
   return     NewProps := merge( ReqProps, '4.Props );
end;
```

Fig. 14: Specification of additionally needed complex productions

Type Scheme of a Graph

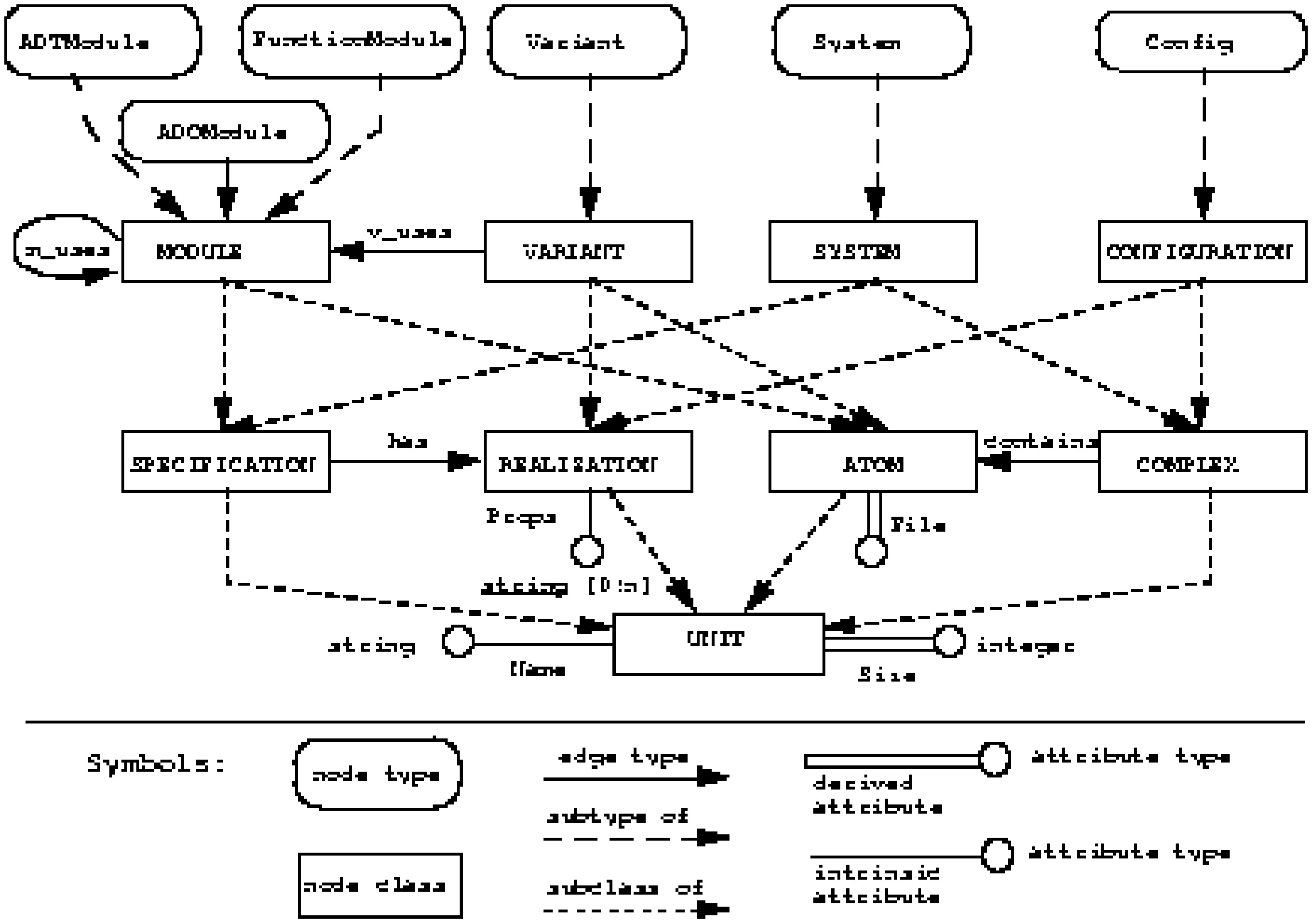


Fig. 5: The graph schema of MIL graphs (without derived relationships)

- *Boxes* with *round corners* represent node types which are connected to their uniquely defined classes by means of *dashed edges* representing "type is instance of class" relationships; the type ADTModule belongs for instance to the class MODULE.
- *Solid edges* between node classes represent edge type definitions; the edge type v_uses is for instance a relationship between VARIANT nodes and MODULE nodes and m_uses edges connect MODULE nodes with other MODULE nodes.
- *Circles* attached to node classes represent attributes with their names above or below