Characterizing Tools for Visual Modeling Techniques

Gabriele Taentzer
Technical University of Berlin, Germany

Abstract
In the wide area of visual modeling techniques a large number of CASE and Meta-CASE tools have been developed to define and work with visual modeling techniques. In this report, we mainly concentrate on those tools developed by partners and grant holders of the Research Training Network SegraVis on Syntactic and Semantic Integration of Visual Modeling Techniques. Each tool is shortly introduced and characterized along a criteria catalog for CASE and MetaCASE tools.

1 Introduction
In the wide area of visual modeling techniques a large number of CASE and Meta-CASE tools have been developed to define and work with visual modeling techniques. In this survey, we concentrate on those tools for visual modeling techniques and languages which have been developed within the SegraVis project. The main purpose of this characterization is to get an overview on the offered functionalities and to better understand each tool’s purpose and features. A characterization of further CASE and Meta-CASE tools, e.g. the large variety of CASE-tools for UML [6], is out of scope of this paper.

A CASE tool is usually dedicated to one individual modeling technique. It supports the editing of models and might offer also support for simulation, validation, transformation and code generation. For example, there are a number of UML-CASE tools such as MagicDraw\(^1\), Poseidon\(^2\), etc.

Meta CASE tools support in specifying visual modeling techniques and generating visual modeling environments. Different kinds of Meta CASE tools are available: Generic, parameterizable CASE tools allow for the definition of variants of one main modeling techniques (e.g. lots of UML CASE tools offer support for the definition of stereotypes). CASE tool frameworks (such as Eclipse/EMF) can be used to generate reusable, semi-complete code to be extended to a specific CASE tool. CASE tool generators offer designers support for the specification of visual modeling environments and their generation from the given specification. Tools like AToM3 and Tiger described below belong to this group of Meta CASE tools.

The Meta CASE approaches followed by the following tools are graph transformation-based and/or based on Meta Object Facilities (MOF) [5]. Basing the specification of a visual modeling technique on graph transformation [8], the visual alphabet, i.e. the symbols and links, are described by type graphs. Graph grammars define the language syntax, and graph transformation systems can be used for the semantics definition. Using MOF, symbols, links

\(^1\) www.magicdraw.com
\(^2\) www.gentleware.com
and multiplicity constraints are described by class diagrams, while well-formedness rules define the language syntax.

The semantics of visual modeling techniques can be described in an operational way by defining a number of transition steps which might be animated. If a separate semantical domain is defined, a visual model has to be translated to that domain. This is usually done by some transformation concept, such as graph transformation or QVT, the OMG-approach. Model transformation has become a central activity in model-driven software development. Model transformations have been classified by Czarnecki, Mens et.al. [2, 4].

We start this tool characterization by introducing each tool first. Thereafter, a catalog of criteria is presented which distinguishes functional and non-functional characteristics. The functional characteristics are further structured along CASE and MetaCASE-functionalities. Each tool presented is characterized along these criteria as far as adequate. For a better overview, the characterization is presented by a number of tables. Please note that this paper is a revised and extended version of a first comparison of SegraVis tools [9].

2 Survey on SegraVis Tools

In the following, CASE and Meta-CASE tools developed within the SegraVis project, are shortly presented.

**AGG** AGG is a development environment for attributed graph transformation systems supporting an algebraic approach to graph transformation. It aims at specifying and rapid prototyping applications with complex, graph structured data. AGG may be (re)used (without GUI) as a general purpose graph transformation engine in Java applications employing graph transformation concepts.

**MOFLON** The MOFLON meta modeling framework, aims to combine the standardized MOF 2.0 meta modeling language as graph schema language with Fujaba graph transformation rules. MOFLON has adopted the MOMoC code generator to produce JMI-compliant Java code. The framework is used to specify tools for tool integration and trend analysis of domain-specific software architectures.

**GenGED** The GenGED approach (Generation of Graphical Environments for Design) supports the generic description of visual modeling languages for the generation of graphical editors and the simulation of behavior models. GenGED is based on algebraic graph transformation, i.e. AGG, and graphical constraint solving techniques and tools. It has been applied to a variety of visual languages (VLs). The corresponding visual environment supports the visual description of VLs and the generation of language-specific graphical editors, available in syntax-directed or free-hand editing mode. The behavior of a visual model can be specified and simulated in the generated graphical editor.

**AToM3** The two main tasks of AToM3 are meta-modeling and model-transforming. Meta-modeling refers to the description, or modeling of different kinds of formalisms used...
to model systems Model-transforming refers to the (automatic) process of converting, translating or modifying a model in a given formalism, into another model that might or might not be in the same formalism. In AToM3, formalisms and models are described as graphs. From a meta-specification (in the ER formalism or class diagrams) of a formalism, AToM3 generates a tool to visually manipulate (create and edit) models described in the specified formalism. Model transformations are performed by graph rewriting. The transformations themselves can thus be declaratively expressed as graph-grammar models.

**Fujaba** The primary topic of the Fujaba Tool Suite project is to provide an easy way to extend UML and Java development platform with the ability to add plug-ins. The Fujaba Tool Suite combines UML class diagrams and UML behaviour diagrams to a powerful, easy to use, yet formal system design and specification language. Furthermore the Fujaba Tool Suite supports the generation of Java source code out of the whole design which results in an executable prototype. Moreover the way back is provided, too (to some extent so far), so that Java source code can be parsed and represented within UML.

The plug-in SPin allows to specify models on a very high level of abstraction and to transform them into a common concrete representation. SPin realizes the idea of Architecture Stratification and extends Fujaba with a model transformation engine and an Open API for defining custom transformation rules.

**MetaEnv** MetaEnv is a toolbox for automating visual software engineering. MetaEnv augments visual diagrammatic (VD) notations with customizable dynamic semantics. Traditional meta-CASE tools support flexibility at syntactic level: MetaEnv augments them with semantic flexibility. MetaEnv refers to a framework based on graph grammars and has been experimented as add-on to several commercial and proprietary tools that support syntactic manipulation of VD notations.

**ViaTra2** The VIATRA²(VIstual Automated model TRAnsformations) framework is the core of a transformation-based verification and validation environment for improving the quality of systems designed using the Unified Modeling Language or various Business Process Modeling languages by automatically checking consistency, completeness and dependability requirements. The specification formalism of VIATRA2 combines graph transformation and abstract state machines into a single semantic framework. On the tool-level, VIATRA2 is integrated to the Eclipse framework as a plug-in, and it is ported to several off-the-shelf modeling CASE tools.

**Consistency Workbench** The Consistency Workbench is a research prototype for consistency management in UML-based development processes. Currently, consistency of UML models is only partially ensured by the language specification. In particular, behavioral consistency of UML models is not prescribed by the language standard. Such semantic consistency must be defined and checked by the software engineer when applying UML in practical development processes.

The Consistency Workbench aims at providing tool support for consistency management along a general methodology. Briefly, the methodology requires the software engineer to identify consistency problems and then develop partial mappings (model transformations) of

---

7 www.fujaba.de
8 www.eclipse.org/gmt
9 wwwcs.uni-paderborn.de/cs/ag-engels/ag_dt/Tools/ConWork
UML models into a formal semantic domain. In such a semantic domain, formal consistency conditions can be defined and existing formal verification tools such as model checkers can be applied for their verification. One key functionality of the Consistency Workbench is the definition and execution of model transformations as well as consistency checks including model transformations.

**UGT**  UGT [10](UML to Graph Transformation) is a research prototype which generates graph transformation rules from a given UML model. UGT reads a model specification from a given text file and automatically generates the graph rules which facilitate a stepwise execution of the model. The initial graph is computed from the input model plus an object diagram specified by the modeler as initial.

Technically, UGT combines two well established tools in order to realize the functionality described above. The graph transformation part of UGT is realized by the graph transformation tool AGG and an extension of the validation tool [7] is used for the evaluation of OCL expressions.

**Tiger**  The TIGER [10] environment (Transformation-induced Generation of Modeling Environments) supports the generation of visual editor plug-ins in ECLIPSE from formal visual language (VL) specifications, based on meta-modeling and graph transformation. Tiger combines graph transformation concepts offered by AGG with sophisticated graphical editor development features offered by the ECLIPSE Graphical Editing Framework (GEF). Editor commands are modeled in a rule-based way using the Tiger Designer component. Tiger extends AGG by a concrete visual syntax definition for flexible means for visual model representation. From the definition of the VL, Java source code is generated which implements an ECLIPSE visual editor plug-in based on GEF offering an efficient and standardized way for graphical layouting.

The Tiger EMF Transformator [11] is an extension for in-place EMF transformation based on graph transformation. Transformations are visually defined by rules on object patterns typed over an EMF core model. A transformation system can be either compiled to Java code or interpreted using the underlying AGG transformation engine.

### 3 Non-functional characteristics

The main non-functional information about the tools is listed in Fig. 1. Please note that the tools are mainly described along qualitative characteristics, since there are nearly no benchmark tests available for them. In the following, all the non-functional characteristics are listed.

- **Name**: the full name of the tool as well as its shortcut (if available).
- **Developer**: For Segravis tools, this is either the name of a SegraVis partner or a grant holder. For all other tools, this is the name of the head of the development or a company/organisation.
- **Status of tool/component development**: Open source or commercial? Under which license? Prototype, established tool, the standard tool for....?
- **Which version has been used for evaluation**?

---

[10] tfs.cs.tu-berlin.de/tigerproj
Which implementation language has been used?
For which platforms is the tool available?
In which way is the tool documented?
Is there support for interoperability with other tools? What kinds of exchange formats are supported? Are there well-defined interfaces for the tool and/or tool components?
What are the possibilities to extend the tool or components of the tool? Can it be adapted to special user requirements?
Do there exist test suites? How is the recovery from failures?
Are there benchmark tests?

4 Functional characteristics

Describing the functional tool characteristics, we distinguish two groups: tools with CASE functionalities (see Fig. 2) and tools with Meta CASE functionalities. The Meta CASE functionalities comprise general aspects as well as syntactical (see Fig. 3) and semantical features (see Fig. 4). Dependent on the features of each tool, it occurs in those tables only where corresponding characteristics are compared.

4.1 CASE functionalities

A variety of considered tools are CASE tools for a fixed language each. A CASE tool should be an integrated development environment for a certain modeling language. It can comprise visual editors, simulators, model transformations to other modeling techniques, code generators, animation and validation tools. The table entries are concerned with the following questions:

Which visual modeling technique or language is supported?
Is there a reference application for this tool? If yes, the one or two most important ones are mentioned.
Is the tool developed for a special application domain?
Is there a visual editor? How do the visual editors work? Syntax-directed or freehand?
Is there a simulator? Is it visual? How does it work? Is the simulation discrete or continuousanimated? Is it hand-driven or automatic? Does it show the (intermediate) results? In a visual form?
To which other visual modeling technique or language are model transformations supported?
To which implementation languages are code generators available?
Which kinds of model validation techniques are supported?
Are several views on the model supported? If yes, which ones?
### Non-functional Characteristics: SeGraVis Tools

<table>
<thead>
<tr>
<th>Name</th>
<th>AGG</th>
<th>MOFLON</th>
<th>GenGED</th>
<th>AToM3</th>
<th>Fujaba</th>
<th>MetaEnv</th>
<th>VIATRA2</th>
<th>Consistency</th>
<th>Workbench</th>
<th>UGT</th>
<th>Tiger</th>
</tr>
</thead>
<tbody>
<tr>
<td>version</td>
<td>1.2.6</td>
<td>0.9</td>
<td>1.0</td>
<td>0.2.2</td>
<td>4.3.2</td>
<td>1.0</td>
<td>2.0 (Release 2)</td>
<td>1.0</td>
<td>0.1</td>
<td>1.2.0</td>
<td></td>
</tr>
<tr>
<td>implementation</td>
<td>Java</td>
<td>Java</td>
<td>Java</td>
<td>Python</td>
<td>Java</td>
<td>C++</td>
<td>Java</td>
<td>Java</td>
<td>Java</td>
<td>Java</td>
<td>Java</td>
</tr>
<tr>
<td>language</td>
<td>Java</td>
<td>SVG</td>
<td>Python</td>
<td>Java</td>
<td>n/a</td>
<td>Java</td>
<td>Java</td>
<td>n/a</td>
<td>n/a</td>
<td>Java</td>
<td>Java</td>
</tr>
<tr>
<td>supported</td>
<td>any, JDK</td>
<td>any, JDK</td>
<td>UNIX, Linux</td>
<td>any, JDK</td>
<td>any, JDK</td>
<td>Windows</td>
<td>JDK, Eclipse</td>
<td>any, JDK</td>
<td>any, JDK</td>
<td>any, JDK</td>
<td>any, JDK</td>
</tr>
<tr>
<td>platforms</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>support for</td>
<td>Transformation engine with API, GXL</td>
<td>XMI, JMI</td>
<td>XML</td>
<td>Integration with Python applications</td>
<td>GXL, XMI</td>
<td>Transform engine with API</td>
<td>Transf. Engine with API, VPML, VCTL, plugins</td>
<td>none</td>
<td>none</td>
<td>any, JDK</td>
<td></td>
</tr>
<tr>
<td>interoperability</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>input formats</td>
<td>GXL, GGX, ECORE</td>
<td>XML, proprietary format</td>
<td>proprietary XML format</td>
<td>AtoM3 format</td>
<td>FPR, CR, proprietary XML</td>
<td>UML models produced by Ptolemy 1.6</td>
<td>UML models in extended USE input language</td>
<td>proprietary XML format; gen. editor: GGX</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>output formats</td>
<td>GXL, GGX, GTXL</td>
<td>XML, proprietary layout format</td>
<td>proprietary XML format</td>
<td>AtoM3 format</td>
<td>FPR, CR, proprietary XML</td>
<td>any textual (code generation)</td>
<td>CSP process algebra description</td>
<td>proprietary XML format; gen. editor: GGX</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>extension</td>
<td>flexible attribution by Java expressions</td>
<td>Plugin mechanism</td>
<td>none</td>
<td>attribute by Python expressions</td>
<td>Plugin mechanism</td>
<td>none</td>
<td>New plugins, external Java programs called during transformation</td>
<td>none</td>
<td>none</td>
<td>none</td>
<td></td>
</tr>
<tr>
<td>possibilities</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reliability</td>
<td>JUnit</td>
<td>JUnit</td>
<td>none</td>
<td>none</td>
<td>JUnit tests</td>
<td>none</td>
<td>none</td>
<td>none</td>
<td>none</td>
<td>none</td>
<td>none</td>
</tr>
<tr>
<td>/ test suites</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Performance /</td>
<td>none</td>
<td>none</td>
<td>none</td>
<td>none</td>
<td>none</td>
<td>none</td>
<td>none</td>
<td>none</td>
<td>none</td>
<td>none</td>
<td>none</td>
</tr>
<tr>
<td>benchmark tests</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 1: SegraVis Tools: Non-functional characteristics
### CASE functionalities: SegraVis Tools

<table>
<thead>
<tr>
<th>Name</th>
<th>AGG</th>
<th>MOFLON</th>
<th>Fujaba</th>
<th>MetaEnv</th>
<th>ConsistencyWorkbench</th>
<th>UGT</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>visual modeling</strong></td>
<td><strong>techniques/language</strong></td>
<td>graph transformation</td>
<td>UML Infrastructure</td>
<td>Story Driven Modelling (graph transformation)</td>
<td>graph transformation</td>
<td>graph transformation and UML-like modelling of consistency checks</td>
</tr>
<tr>
<td><strong>reference applications</strong></td>
<td></td>
<td>Generation of visual editors</td>
<td>none</td>
<td>MDA and embedded system</td>
<td>PLCTools</td>
<td>consistency checking of UML statecharts by translation into semantic domain CSP</td>
</tr>
<tr>
<td><strong>special application domains</strong></td>
<td></td>
<td>any</td>
<td>embedded systems</td>
<td>any</td>
<td>any</td>
<td>any</td>
</tr>
<tr>
<td><strong>visual model editors</strong></td>
<td>syntax-directed</td>
<td>syntax-directed</td>
<td>syntax-directed</td>
<td>none</td>
<td>syntax-directed</td>
<td>none</td>
</tr>
<tr>
<td><strong>simulators</strong></td>
<td>discrete, hand-driven/automatic</td>
<td>none</td>
<td>discrete, graph browser</td>
<td>none</td>
<td></td>
<td>discrete, hand-driven, visual, shows intermediate results</td>
</tr>
<tr>
<td><strong>model transformations to</strong></td>
<td>any</td>
<td>any</td>
<td>any</td>
<td>Petri nets</td>
<td>semantic domain</td>
<td>graph transformation system</td>
</tr>
<tr>
<td><strong>code generators to</strong></td>
<td>none</td>
<td>Java</td>
<td>Java</td>
<td>ANSI C</td>
<td>none</td>
<td>none</td>
</tr>
<tr>
<td><strong>model validation</strong></td>
<td>graph parsing, consistency checking, critical pair analysis, termination</td>
<td>none</td>
<td>none</td>
<td>benchmarks</td>
<td>consistency checking by executing model transformations and performing model checking in semantic domain</td>
<td>validation by simulating system runs by transforming system state graphs</td>
</tr>
<tr>
<td><strong>several views</strong></td>
<td>graph view w/o node icons, attributes</td>
<td>no</td>
<td>view diagrams</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
</tbody>
</table>

**Figure 2: SegraVis Tools: CASE functionalities**
4.2 Meta CASE functionalities

For each Meta CASE tool, we consider the scope of the tool: For which kinds of modeling techniques and/or languages is this Meta CASE tool designed? Moreover, the approach for defining the visual modeling technique/language is interesting. Which one is used?

We characterize Meta CASE tools along the supporting tools offered for each language definition aspect. Language definition aspects are the syntax definition, simulation and animation aspects, and model transformation. For each aspect, we distinguish tools to define this aspect, tools interpreting this definition, tools generated from a definition, and tools analyzing a definition according to certain properties.

General and Syntax Aspects

- Which kinds of modeling techniques/languages can be described by this tool?
- What are the meta concepts to describe a visual technique or language? If several are used, please distinguish which one is used for which purpose.
- Is there a reference application for this tool? If yes, the one or two most important ones are mentioned.
- Is the tool developed for a special application domain?
- Are abstract and/or concrete syntax features defined?
- How are symbols (model elements) and their interrelations defined?
- Which structures based on symbols and relations are allowed, i.e. belong to the visual language to be defined? Are all structures allowed? Or are they restricted by additional constraints or a grammar?
- Which kind of concrete layout is possible? Graph-like, diagram-like, icon-based, etc.?
- If the concrete syntax is described, how is the concrete layout defined? Are special layout algorithms used?
- Is there a visual editor which takes the language description as input and interprets it such that an editor for the language defined is available? Which features has this editor? (See CASE tool simulators)
- Is it possible to generate a visual editor for the language defined? Which features has this editor? (See CASE tool simulators)
- Is parsing of visual structures/models supported? Are certain parts, i.e. texts parsed?
- Is it possible to formulate syntactical constraints? Which kinds of constraints can be used?

Semantics Aspects

- Is it possible to give an operational semantics for the language to be defined?
- Is it possible to translate language elements to some separate semantical domain? Which one?
- How is the semantics described?
- Is there a simulator which takes the language description as input and interprets it such that a simulator for the language defined is available? Which features has this simulator? (See CASE tool simulators)
<table>
<thead>
<tr>
<th>Name</th>
<th>AGG</th>
<th>MOFLON</th>
<th>GenGED</th>
<th>CD-VML-UC</th>
<th>AToM3</th>
<th>Fujaba</th>
<th>MetaEnv</th>
<th>VIATRA2</th>
<th>Consistency Workbench</th>
<th>Tiger</th>
</tr>
</thead>
<tbody>
<tr>
<td>supported modeling</td>
<td>any</td>
<td>MOF 2.0</td>
<td>UML 2.0</td>
<td>any</td>
<td>UML-like</td>
<td>any</td>
<td>any</td>
<td>any</td>
<td>any</td>
<td>any</td>
</tr>
<tr>
<td>languages/techniques</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>approach for language</td>
<td>typed-attributed graph transformation</td>
<td>graph transformation</td>
<td>typed-attributed graph structures</td>
<td>meta modeling</td>
<td>Meta modeling for syntax, graph transformation for semantics</td>
<td>Object oriented graph transformation</td>
<td>typed graph transformation</td>
<td>VPM metamodeling, graph transformation, ASMs</td>
<td>graph transformation rules and activity-diagram like for consistency checks</td>
<td>typed attributed graph transformation</td>
</tr>
<tr>
<td>definition</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>reference applications</td>
<td>Petri nets, statecharts, activity diagrams</td>
<td>bootstrap</td>
<td>Petri nets, Statecharts</td>
<td>Timed Automata, Process Interaction, DEVS</td>
<td>MOF 2.0</td>
<td>PTCTools</td>
<td>Dependability evaluation of UML and BPM/BPEL models</td>
<td>consistency checking of UML statecharts by translation into semantic domain CSP</td>
<td>Petri nets, automata, activity diagrams; model transformation from activity diagrams to Petri nets, refactoring of EMF models</td>
<td></td>
</tr>
<tr>
<td>special application</td>
<td>none</td>
<td>tool integration, re-engineering</td>
<td>visual environment generation</td>
<td>(multi-formalism) simulation</td>
<td>Tool integration, re-engineering, real-time sys</td>
<td>model transformation</td>
<td>model transformation</td>
<td>consistency checking of models</td>
<td>visual environment generation</td>
<td></td>
</tr>
<tr>
<td>domains</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>syntax features</td>
<td>abstract syntax</td>
<td>any</td>
<td>concrete and abstract syntax</td>
<td>concrete and abstract syntax</td>
<td>concrete and abstract syntax</td>
<td>abstract syntax</td>
<td>abstract syntax (in GEF)</td>
<td>concrete and abstract syntax</td>
<td></td>
<td></td>
</tr>
<tr>
<td>symbols and relations</td>
<td>typed, attributed nodes and edges</td>
<td>classes, associations, mutual references</td>
<td>typed, attributed symbols and relations</td>
<td>model elements mechanisms</td>
<td>typed, attributed symbols and relations</td>
<td>classes and associations</td>
<td>typed, attributed nodes and edges</td>
<td>classes (meta) models, associations, attributes</td>
<td>typed, attributed symbols and relations</td>
<td></td>
</tr>
<tr>
<td>allowed structures</td>
<td>type graph with node type inheritance</td>
<td>MOF package structure</td>
<td>defined by graph grammar</td>
<td>n/a</td>
<td>defined by meta-modelling</td>
<td>defined by an attributed type graph</td>
<td>class/object diagrams</td>
<td>type graph with inheritance and syntax graph grammar</td>
<td></td>
<td></td>
</tr>
<tr>
<td>concrete layout</td>
<td>graph-like</td>
<td>graph-like</td>
<td>diagrammatic or icon-based</td>
<td>n/a</td>
<td>icon-based, no algorithm for layout</td>
<td>graph-like</td>
<td>graph-like</td>
<td>Integrated tree view based + graphical model viewer</td>
<td>graph-like, based on Eclipse Draw2D Figures</td>
<td></td>
</tr>
<tr>
<td>Layout description</td>
<td>node and edge graphs</td>
<td>node and edge types, type graphs</td>
<td>visual alphabet and grammar + editing rules</td>
<td>meta-model (constraints in Python)</td>
<td>node and edge graphs, type graphs</td>
<td>Built-in layout algorithms for model viewer</td>
<td>visual alphabet, based on Eclipse GEF constraints</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>interpreting editor</td>
<td>syntax-directed</td>
<td>none</td>
<td>none</td>
<td>combination free hand / syntax directed</td>
<td>none</td>
<td>none</td>
<td>none</td>
<td>none</td>
<td>none</td>
<td></td>
</tr>
<tr>
<td>generated editor</td>
<td>none</td>
<td>yes</td>
<td>none</td>
<td>yes</td>
<td>yes</td>
<td>none</td>
<td>none</td>
<td>none</td>
<td>none</td>
<td></td>
</tr>
<tr>
<td>parsing</td>
<td>For textual format</td>
<td>Java</td>
<td>For textual format</td>
<td>none</td>
<td>Java</td>
<td>none</td>
<td>VTCL (textual format)</td>
<td>none</td>
<td>none</td>
<td></td>
</tr>
<tr>
<td>syntactical constraints</td>
<td>OCL constraints</td>
<td>Meta constraints</td>
<td>Constraints</td>
<td>yes</td>
<td>Class diagram</td>
<td>none</td>
<td>Metamodel inheritance, instantiation, (restricted) multiplicities</td>
<td>attributed type graph with inheritance, syntax graph grammar</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
• Is it possible to generate a simulator for the language defined? Which features has this simulator? (See CASE tool simulators)
• Are animated simulations supported?
• If animation is supported, which animation features are available? Continuous movements, color changes, size changes, change of visibilities,...?
• How can the model behaviour be tested? Are test cases generated?
• Is the validation of model behaviour supported? If yes, how can it be validated?

Transformation and Integration Aspects
• If model transformation is supported, which target models or target languages are possible?
• How is the model transformation described?
• Is there an interpreter which takes a model transformation description as input and interprets it?
• Is it possible to generate code for a model transformation? In which language?
• Is it possible to animate a model transformation?
• Is it possible to validate a model transformation? If yes, what kinds of validations can be performed?
• Is it possible to integrate several languages?
• Is there some consistency checking between different models or languages available?

5 Survey on Additional Tools for Visual Modeling Techniques

In the following, two further tools for visual modeling techniques are presented. These tools are presented by participants of the “Advanced School on Visual Modeling techniques” which was held in Leicester, September 9-11, 2006.

GROOVE The GROOVE Tool Set consists of a (graph) Editor and a (graph transformation) Simulator. Its main purpose is to support the generation of transition systems that stem from graph production systems. It then allows to perform verification techniques (e.g. model checking) on these transition systems, in which the states are represented as graphs taking graph structures as building blocks for the properties to check for. In the future it is planned to implement partial order reduction techniques as well as abstraction techniques enabling verification of large (or even infinite) systems.

GROOVE, version 1.4.0, has been developed at the University of Twente by A. Rensink, H. Kastenberg, and T. Staijen. It is free software (GNU Public License) implemented in Java. It is documented by papers and example production systems. A separate transformation engine with API is available. GXL is used as input/output format. This has been tested using JUnit.

GROOVE can be used as CASE tool for graph transformation. The reference application is state space exploration/verification. The tool environment comprises a syntax-directed
<table>
<thead>
<tr>
<th>Name</th>
<th>AGG</th>
<th>MOFLON</th>
<th>GenGED</th>
<th>AToM3</th>
<th>Fujaba</th>
<th>MetaEnv</th>
<th>VIATRA2</th>
<th>Consistency Workbench</th>
<th>Tiger</th>
</tr>
</thead>
<tbody>
<tr>
<td>operational semantics</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>for consistency checks</td>
<td>yes</td>
</tr>
<tr>
<td>semantical domain</td>
<td>-</td>
<td>any</td>
<td>-</td>
<td>any</td>
<td>any</td>
<td>Petri nets</td>
<td>ASM</td>
<td>-</td>
<td>any</td>
</tr>
<tr>
<td>semantics description</td>
<td>graph transformation system</td>
<td>graph transformation system</td>
<td>graph transformation system</td>
<td>graph transformation system</td>
<td>graph transformation system</td>
<td>consistency checks are specified as activity diagrams</td>
<td>graph transformation system</td>
<td>graph transformation system</td>
<td></td>
</tr>
<tr>
<td>interpreting simulator</td>
<td>visual, discrete, interactive/ automatic</td>
<td>none</td>
<td>visual, discreet/ continuous, interactive/ automatic</td>
<td>discrete/ continuous, interactive/ automatic</td>
<td>Dynamic Object Browsing System (DOBS)</td>
<td>by means of token game</td>
<td>discrete sim. with visual model browser</td>
<td>for consistency checks</td>
<td>interpreter of EMF rules in AGG</td>
</tr>
<tr>
<td>generated simulator</td>
<td>none</td>
<td>none</td>
<td>none</td>
<td>none</td>
<td>yes</td>
<td>none</td>
<td>none</td>
<td>discrete simulation by simulation rules</td>
<td></td>
</tr>
<tr>
<td>animation views</td>
<td>none</td>
<td>none</td>
<td>yes, 2D</td>
<td>yes</td>
<td>none</td>
<td>in the target editor</td>
<td>none</td>
<td>none</td>
<td></td>
</tr>
<tr>
<td>animation features</td>
<td>none</td>
<td>none</td>
<td>continuous movement, color/size/visibility changes</td>
<td>yes</td>
<td>none</td>
<td>Animation of the Petri net token game</td>
<td>none</td>
<td>none</td>
<td></td>
</tr>
<tr>
<td>Testing of model dynamics</td>
<td>Rule applications, no generation of test cases</td>
<td>JUnit</td>
<td>none</td>
<td>Rule applications</td>
<td>JUnit</td>
<td>none</td>
<td>By interactive simulation</td>
<td>Rule application, no generation of test cases</td>
<td></td>
</tr>
<tr>
<td>validation of model dynamics</td>
<td>conflicts and dependencies, reachability of graphs, termination</td>
<td>incremental analysis</td>
<td>none</td>
<td>simple model checker</td>
<td>none</td>
<td>benchmarks</td>
<td>none</td>
<td>same as for AGG</td>
<td></td>
</tr>
</tbody>
</table>
### Meta CASE functionalities – transformation and integration aspects: SegraVis Tools

<table>
<thead>
<tr>
<th>Name</th>
<th>MOFLON</th>
<th>GenGED</th>
<th>AToM3</th>
<th>MetaEnv</th>
<th>Viatra2</th>
<th>ConsistencyWorkbench</th>
<th>Tiger</th>
</tr>
</thead>
<tbody>
<tr>
<td>target models/languages</td>
<td>MOF 2.0, MOF 2.x</td>
<td>Petri nets</td>
<td>Any</td>
<td>High-level Timed Petri nets</td>
<td>any</td>
<td>user-defined</td>
<td>any</td>
</tr>
<tr>
<td>Transformation description</td>
<td>graph transformation</td>
<td>XML Stylesheets</td>
<td>graph transformation</td>
<td>graph transformation</td>
<td>graph transformation</td>
<td>model transformation system</td>
<td>graph transformation</td>
</tr>
<tr>
<td>interpreted model transformation</td>
<td>no</td>
<td>yes, by integrating the alphabets</td>
<td>yes</td>
<td>Petri net models</td>
<td>yes (also code generation)</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>generated model transformation</td>
<td>Java</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>for Java</td>
<td>no</td>
<td>Java</td>
</tr>
<tr>
<td>animation of transformation</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>by means of token game</td>
<td>no</td>
<td>partially</td>
<td>no</td>
</tr>
<tr>
<td>validation of transformation</td>
<td>incremental analysis</td>
<td>no</td>
<td>No</td>
<td>benchmarks</td>
<td>no</td>
<td>partially</td>
<td>conflicts and dependencies, termination</td>
</tr>
<tr>
<td>integration of different languages</td>
<td>no</td>
<td>yes, by integrating the alphabets</td>
<td>yes, several meta-models</td>
<td>through a common semantic domain</td>
<td>by metamodel integration</td>
<td>yes</td>
<td>yes, by integrating the alphabets</td>
</tr>
<tr>
<td>consistency checking between different models/languages</td>
<td>yes</td>
<td>none</td>
<td>none</td>
<td>none</td>
<td>none</td>
<td>yes</td>
<td>none</td>
</tr>
</tbody>
</table>
editor, a discrete, hand-driven or automatic simulator which keeps track of intermediate
results, several views on state graph, rules, and transition system, as well as a tool for model
transformation.

Using GROOVE as MetaCASE tool, it is especially useful to define the abstract syntax
and semantics of object-oriented systems by graph transformation. An interpreting simulator
is available which is visual, discrete, and hand-driven or automatic. The exploration of
the state space is animated by color changes. The model behaviour can be tested by rule
applications and validated by a simple model checker.

TEMA  The TEMA (Test Modelling using Action words) tool is created for model-based
testing Symbian S60 devices. The tool contains a visual editor for creating test models and
routines for running the tests. The test models are composed of model components which
are presented as LSTs, that is, labeled transition systems (LTSs) where also states can
be labelled. Composing a test model means here a process algebraic parallel composition
of model components which have gone through a set of small transformations. Parallel
composition synchronisations, transformations and the specified form of LSTS have been
tuned to support creating models especially for Symbian testing.

TEMA has been developed by Antti Kervinen at the Tampere University of Technology
and is a tool in a very early state which will be open source later. It is implemented in Java
and Python and runs on Linux, MacOS, Unix, Windows for modelling. The test adapter
for Symbian devices is Windows only. TEMA can interoperate with Mercury QTP (test
adapter), Tampere Verification Tool and understands LSTS and CSV as input formats. The
output is just text.

TEMA is a CASE tool for LSTS and restricted process algebras and is applied for model-
based testing tools. A special application domain is the testing of Symbian S60 devices.
The tool has visual editors where LTS/LSTS may be drawn free-hand, hand-driven and
automatic simulators. The models can be checked concerning sanity checks. Moreover,
external verification tools can be used.

6 Conclusion

In this contribution, we presented CASE and Meta CASE tools for visual modeling tech-
niques and characterized them according to a criteria catalogue. The main purpose of this
characterization is to get an overview on the functionalities of such kind of tools and to find
out which features are already covered by tools.

Although CASE functionalities are discussed within this paper, the main focus lies on
MetaCASE functionalities. In a first conclusion, we can say that the syntax definition as well
as model transformation aspects are quite well captured by the tools presented. These VL
aspects are defined based on MOF and graph transformation. Interpreters and generators
are available to provide the user of a visual modeling technique with visual editors and tool
support for model transformation. Moreover, several analysis and verification techniques for
VL's are available. Modularity, refinement and integration is not yet covered to such extent
by the tools considered.

A characterization and comparison with other CASE and Meta-CASE tools is left to
future work.
Acknowledgement

I would like to thank the developers of all tools presented in this paper for their help in tool characterization.

References


