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 wich 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[3] 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[4] 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[5] 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[6] are meta-modeling and model-transforming. Meta-modeling refers to the description, or modeling of different kinds of formalisms used

[3] `tfs.cs.tu-berlin.de/agg`
[4] `www.moflon.org`
[5] `tfs.cs.tu-berlin.de/genged`
[6] `atom3.cs.mcgill.ca`

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[7] 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 [3] 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 [1] 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[8] (VIsual 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[9] 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/tigerprj`
[11] `tfs.cs.tu-berlin.de/emftrans`

- 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 continous/animated? 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

Name	AGG	MOFLON	GenGED	AToM3	Fujaba	MetaEnv	VIATRA2	Consistency Workbench	UGT	Tiger
developer	TU Berlin	TU Darmstadt	TU Berlin	Juan de Lara (Hans Vangheluwe)	Uni Paderborn (Uni Kassel)	Politecnico di Milano	Budapest University of Technology and Economics	Uni Paderborn	Uni Bremen	TU Berlin
status of tool/component development	free software, GNU License, established	free software	free software, established	free software	free software, GNU LGPL, established	free software	free software available under EPL	research prototype, currently no further development	research prototype, active, not yet released	free software
version	1.2.6	0.9	1.0	0.2.2	4.3.2	1.0	2.0 (Release 2)	1.0	0.1	1.2.0
implementation language	Java	Java	Java	Python	Java	C++	Java	Java	Java	Java
implementation language of generated tools	n/a	Java	SVG	Python	Java	n/a	Java	n/a	n/a	Java
supported platforms	any, JDK	any, JDK	UNIX, Linux	UNIX, Linux, Windows, MacOS	any JDK	Windows	JDK, Eclipse	any, JDK	any, JDK	any, JDK , Eclipse 3.2
documentation	paper, Web pages	papers, Web pages	papers, book, Web pages	Several tutorials on the web	papers, Javadoc for API, wiki	paper	User guide with various examples (incomplete)	available as Technical Report TR-RI-03-245	diploma thesis	papers, web pages, Javadoc
support for interoperability	Transformation engine with API, GXL	XMI, JMI	XML	integration with Python applications	GXL, XMI	Transformation engine with API	Transf. Engine with API, VPML, VCTL, plug-ins	none	none	XML, GGX (integration with AGG)
input formats	GXL, GGX, ECORE	XMI, proprietary layout format	proprietary XML format	AtoM3 format	FPR, CXR, proprietary XML	proprietary	VPML, VTCL, UML2, BPEL, WSDL, EMF (alpha)	UML models produced by Poseidon 1.6	UML models in extended USE input language	proprietary XML format; gen. editor: GGX
output formats	GXL, GGX, GTXL	XMI, proprietary layout format	proprietary XML format, SVG	AtoM3 format, GGX, OOCSMP, GPSS, PNS	FPR, CXR, proprietary XML	proprietary	any textual (code generation)	CSP process algebra description	GGX	proprietary XML format; gen. editor: GGX
extension possibilities	flexible attribution by Java expressions	Plugin mechanism	none	attribution by Python expressions	Plugin mechanism	none	New plugins, external Java programs called during transformation	none	none	attribution by Java expr; Eclipse plugin mechnism
Reliability / test suites	Junit	JUnit	none	none	JUnit tests	none	no	none	none	none
Performance / benchmark tests	none	none	none	none	none	none	none	none	none	none

Figure 1: SegraVis Tools: Non-functional characteristics

CASE functionalities: SegraVis Tools

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

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)

Meta CASE functionalities – general and syntax aspects: SegraVis Tools

Name	AGG	MOFLON	GenGED	CD-VML-UC	AToM3	Fujaba	MetaEnv	ViATRA2	Consistency Workbench	Tiger
supported modeling languages/techniques	any	MOF 2.0, UML 2.0	any	UML-like	Graph-like	UML-like with graph transformations	any	any (metamodel based)	any	any
approach forlanguage definition	typed attributed graph transformation	graph transformation	typed attributed graph structures	meta modeling	Meta modeling for syntax, graph transformation for semantics	Object oriented graph transformation	typed graph transformation	VPM metamodeling, graph transformation, ASMs	graph transformation rules and activity-diagram like description for consistency checks	typed attributed graph transformation
reference applications	Petri nets, statecharts, activity diagrams	bootstrap	Petri nets, Statecharts		Timed Automata, Process Interaction, DEVS, ...	MOF 2.0	PLCTools	Dependability evaluation of UML and BPM/BPEL models	consistency checking of UML statecharts by translation into semantic domain CSP	Petri nets, automata, activity diagrams; model transformation from activity diagrams to Petri nets, refactoring of EMF models
special application domains	none	tool integration, reengineering	visual environment generation		(multi-formalism) simulation	Tool integration, re-engineering, real time sys.	model transformation	model transformation	consistency checking of models	visual environment generation
syntax features	abstract syntax	any	concrete and abstract syntax	concrete/ abstract syntax	concrete and abstract syntax	concrete/ abstract syntax	abstract syntax	abstract syntax (in GEF)		concrete and abstract syntax
symbols and relations	typed, attributed nodes and edges	classes, associations, mutual references	typed, attributed symbols and relations	model elements/ mechanisms	typed, attributed symbols and relations	classes and associations	typed, attributed nodes and edges	classes/(meta)models, associations, attributes		typed, attributed symbols and relations
allowed structures	type graph with node type inheritance	MOF package structure	defined by graph grammar	n/a	defined by meta-modelling	compatible with UML class and package diagrams	defined by an attributed type graph	class/object diagrams		type graph with inheritance and syntax graph grammar
concrete layout	graph-like	graph-like	diagrammatic or icon-based		icon-based, no algorith for layout	graph-like	graph-like	Integrated tree view based + graphical model viewer/ editor		graph-like, based on Eclipse Draw2D-Figures
Layout description	node and edge types, type graphs	node and edge types, type graphs	visual alphabet and grammar + editing rules	Notation element/visual mechanisms	meta-model (+constraints in Python)	graph transformation system	node and edge types, type graphs	Built-in layout algorithms for model viewer		visual alphabet, based on Eclipse GEF constraints
interpreting editor	syntax-directed	none	syntax-directed		combination free hand / syntax directed	none	none	none	none	none
generated editor	none	yes	none		yes	yes	none	none	none	as Eclipse plug-in
parsing	For textual format	Java	For textual format		none	Java	none	VTCL (textual format)		none
syntactical constraints	Graph constraints	OCL constraints		Constraints	yes	Class diagram	none	metamodel inheritance, instantiation, (restricted) multiplicities		attributed type graph with inheritance, syntax grammar

Figure 3: SegraVis Tools: Meta-CASE functionalities: general and syntactical aspects

- 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[12] 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

[12] `groove.sf.net/`

Meta CASE functionalities – semantics aspects: SegraVis Tools

Name	AGG	MOFLON	GenGED	AToM3	Fujaba	MetaEnv	VIATRA2	Consistency Workbench	Tiger
operational semantics	yes	yes	yes	yes	yes	no	yes	for consistency checks	yes
semantical domain	-	any	-	any	any	Petri nets	ASM	-	any
semantics description by	graph transformation system	graph transformation system	graph transformation system	graph transformation system	graph transformation system	graph transformation system	graph transformation + ASMs	consistency checks are specified as activity diagrams	graph transformation system
interpreting simulator	visual, discrete, interactive/ automatic	none	visual, discrete/ continuous, interactive/ automatic	discrete/ continuous, interactive/ automatic	Dynamic Object Browsing System (DOBS)	by means of token game	discrete sim. with visual model browser	for consistency checks	interpreter of EMF rules in AGG
generated simulator	none	none	for animation scenarios		yes	none			discrete simulation by simulation rules
animation views	none	none	yes, 2D	yes	none	in the target editor	none		none
animation features	none	none	continuous movement, color/size/ visibility changes	yes	none	Animation of the Petri net token game	none		none
Testing of model dynamics	Rule applications, no generation of test cases	JUnit	none	Rule applications	JUnit	none	By interactive simulation		Rule application, no generation of test cases
validation of model dynamics	conflicts and dependencies, reachability of graphs, termination	incremental analysis	none	simple model checker	none	benchmarks	none		same as for AGG

Figure 4: SegraVis Tools: Meta-CASE functionalities: semantical aspects

Meta CASE functionalities – transformation and integration aspects: SegraVis Tools

Name	MOFLON	GenGED	AToM3	MetaEnv	Viatra2	ConsistencyWorkbench	Tiger
target models/languages	MOF 2.0, MOF 2.x	Petri nets	Any	High-level Timed Petri nets	any	user-defined	any
Transformation description	graph transformation	XML Stylesheet	graph transformation	graph transformation	graph transformation	model transformation system	graph transformation
interpreted model transformation	no	yes, by integrating the alphabets	yes	Petri net models	yes (also code generation)	yes	yes
generated model transformation	Java	no	no		for Java	no	Java
animation of transformation	no	no	no	by means of token game	no	partially	no
validation of transformation	incremental analysis	no	No	benchmarks	no	partially	conflicts and dependencies, termination
integration of different languages	no	yes, by integrating the alphabets	yes, several meta-models	through a common semantic domain	by metamodel integration	yes	yes, by integrating the alphabets
consistency checking between different models/languages	yes	none	none	none		yes	none

Figure 5: SegraVis Tools: Meta-CASE functionalities: transformation and integration aspects

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 LSTSs, 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 techniques 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

[1] L. Baresi and M. Pezze. Formal interpreters for diagram notations. *ACM Transactions on Software Engineering and Methodology - ACM Press*, 14(1):42 – 84, January 2005.

[2] K. Czarnecki and S. Helsen. Classification of model transformation approaches. In *Online Proc. of the 2nd Workshop on Generative Techniques in the context of Model-Driven Architecture*, Anaheim, October 2003.

[3] T. Khne, M. Girschick, and F. Klar. Tool support for architecture stratification. In H.C. Mayr and R. Breu, editors, *GI (pub.), Proceedings of the Modellierung 2006, Innsbruck, Tirol, Austria*, volume 82 of *LNI*, pages 213–222, 2006.

[4] T. Mens and P. Van Gorp. A Taxonomy of Model Transformation. In *Proc. International Workshop on Graph and Model Transformation (GraMoT'05)*, number 152 in Electronic Notes in Theoretical Computer Science, Tallinn, Estonia, 2006. Elsevier Science.

[5] Object Management Group (OMG). *OMG's MetaObjectFacility*, 2006. URL: `http://www.omg.org/mof`.

[6] Object Management Group (OMG). *Unified Modeling Language - UML Resource Page*, 2006. URL: `http://www.uml.org`.

[7] Mark Richters. *A Precise Approach to Validating UML Models and OCL Constraints*. PhD thesis, Universität Bremen, Logos Verlag, Berlin, BISS Monographs, No. 14, 2002.

[8] G. Rozenberg, editor. *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations*. World Scientific, 1997.

[9] G. Taentzer. A First Comparison of SegraVis Tools. *EASST Newsletter*, 10:12–23, 2005.

[10] Paul Ziemann, Karsten Hölscher, and Martin Gogolla. From UML Models to Graph Transformation Systems. In Mark Minas, editor, *Proceedings of the Workshop on Visual Languages and Formal Methods (VLFM 2004)*, volume 127(4) of *Electronic Notes in Theoretical Computer Science*, pages 17–33. Elsevier Science, 2005.