The Column on Visual Modelling Techniques

News on the SegraVis Research Training Network

Edited by Reiko Heckel *
*University of Leicester, UK

Abstract. *In this third Column on Visual Modeling Techniques we present a first comparison of tools developed by partners and grant holders of the Research Training Network SegraVis on* Syntactic and Semantic Integration of Visual Modeling Techniques.

Keywords: visual modeling techniques, tools

1 A First Comparison of Segravis Tools

In the wide area of visual modelling techniques a large number of CASE and Meta-CASE tools have been developed to define and work with visual modelling techniques. In this survey, we concentrate on those tools for visual modelling techniques and languages which have been developed within the SegraVis project. The main purpose of this first comparison is to get an overview on the functionalities of such kind of tools and to better understand each tool's purpose and features. A comparison with other CASE and Meta-CASE tools is left to future work.

First the tools regarded in this comparison are shortly introduced. Thereafter, we start a first comparison by explaining the items used for comparison. Throughout this survey, functional and non-functional characteristics are distinguished and listed in several tables. In the following, explanations to each table entry can be found.

1.1 Survey on SegraVis Tools

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

Meta CASE tools support for specifying visual modeling techniques and generating visual modeling environments. Different kinds of Meta CASE tools are available: Generic, parameterizable CASE tools allow 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 for the specification of visual modeling environments and their generation from the given specification. Tools like GenGED and AToM3 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). Basing the specification of a visual modeling techniques on graph transformation 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 and multiplicity constraints are described by class diagrams, while well-formedness rules define the language syntax.

AGG AGG (`tfs.cs.tu-berlin.de/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.

DaMoRe

GenGED The GenGED approach (Generation of Graphical Environments for Design)(`tfs.cs.tu-berlin.de/genged`) 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 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 (`atom3.cs.mcgill.ca`) are meta-modelling and model-transforming. Meta-modelling refers to the description, or modelling 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) 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 (`wwwcs.uni-paderborn.de/cs/fujaba`) project is to provide an easy 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, ideally. Moreover the way back is provided, too (to some extend so far), so that Java source code can be parsed and represented within UML.

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 VIATRA (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 by automatically checking consistency, completeness, and dependability requirements.

Consistency Workbench The Consistency Workbench (`wwwcs.uni-paderborn.de/cs/ag-engels/ag_engl/People/Kuester/ResearchTools.html`) 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 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

1.2 Non-functional characteristics

Here the main non-functional information about the tools is listed. Please note that the tools are compared with each other along qualitative characteristics only, since there are not yet benchmark tests available for them.

- 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?
- 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?

1.3 Functional characteristics

These characteristics can also be seen as features. They describe the purpose of the tool. Moreover, we consider here the scope of the tool, i.e. which kinds of visual modeling techniques this tool is developed for. The functional characteristics are structured into two groups: one for CASE functionalities and one for Meta CASE functionalities. The Meta CASE functionalities comprise general aspects as well as syntactical and semantical features.

Dependent on the features of each tool, it occurs in those tables where corresponding characteristics are offered, only.

1.3.1 CASE functionalities

Most of the considered tools are CASE tools for a fixed language each. A CASE tool can comprise visual editors, simulators, model transformations to other modelling techniques, code generators, and animation 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?

Non-functional Characteristics: SeGravis Tools

Name	AGG	DaMoRE	GenGED	AToM3	Fujaba	MetaEnv	Viatra2	Consistency Workbench	UGT
developer	TUB	TUD	TUB	Juan de Lara	UPB	UniMiB	Daniel Varro	UPB	Universität Bremen, AG Datenbank-systeme
status of tool/component development	free software, GNU License, established	active	free software, established	free software	active	free software	experimental after major reengineering (from Prolog to Java platform)	research prototype, currently no further development	research prototype, active, not yet released
version	1.2.6	0.1	1.0	0.2.2	4.0.1	1.0	2.0 (refers to the new platform)	1.0	0.1
implementation language	Java	Java	Java	Python	Java	C++	Java	Java	Java
implementation language of generated tools	n/a	Java	SVG	Python	Java		Java	n/a	n/a
supported platforms	UNIX, Linux, Windows, MacOS	any JDK 1.4	UNIX, Linux	UNIX, Linux, Windows, MacOS	any JDK 1.4	Windows	Java, Eclipse	Java	any JDK 1.4
documentation	paper, Web pages	partially high	papers, book, Web pages	Several tutorials on the web	poor	paper	several papers on concepts, Javadoc for APIs	available as Technical Report TR-RI-03-245	diploma thesis (not yet published)
support for interoperability	Transformation engine with API, GXL	XMI, JMI	XML	integration with Python applications	GXL	Transformation engine with API			none
input formats	GXL, GGX,ECORE	XMI, proprietary layout format	proprietary XML format	AtoM3 format	FPR	proprietary	textual, VPML, EMF (alpha), Rose model (alpha)	UML models produced by Poseidon 1.6	UML models in extended USE input language
output formats	GXL, GGX, GTXL	XMI, proprietary layout format	proprietary XML format, SVG	AtoM3 format, GGX, OOCSMP, GPSS, PNS	FPR	proprietary	any textual (code generation)	CSP process algebra description	GGX
extension possibilities	flexible attribution by Java expressions	Plugin mechanism		attribution by Python expresions	Plugin mechanism	none	yes	None	none
Reliability / test suites	Junit	JUnit	none	none	JUnit tests	none	no	None	none
Performance / benchmark tests	none	no	none	none	no	none	transformations for IBM Business Integrator	None	none

Figure 1: Segravis Tools: Non-functional characteristics

- 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?

1.3.2 Meta CASE functionalities

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

We compare Meta CASE tools along the supporting tools offered for each language aspect. For each aspect we distinguish tools to describe this aspect, tools interpreting this description, tools generated from a description, and tools analysing a description 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 and their interrelations defined?
- Which structures based on symbols and relations are allowed, i.e. belong of 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,....?
- If the concrete syntax is described, how is the 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)

CASE functionalities: SEGRAVIS Tools

Name	AGG	DaMoRE	Fujaba	MetaEnv	ConsistencyWorkbench	UGT
visual modeling techniques/language	graph transformation	UML Infrastructure	Story Driven Modelling (graph transformation)	graph transformation	graph transformation and UML-like modeling of consistency checks	UML and graph transformation
reference applications	Generation of visual editors	no	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
visual model editors	syntax-directed	syntax-directed	syntax-directed	none	syntax-directed	none
simulators	discrete, hand-driven/automatic	no	yes, graph transformation and 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, Eiffel?, C?, C++?	ANSI C		none
model validation	graph parsing, consistency checking, critical pair analysis	no	no	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	none		no

Figure 2: Segravis Tools: CASE functionalities

- 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?

- 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)
- 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?

1.4 Conclusion

In this contribution, we presented CASE and Meta CASE tools developed within the Segravis project and started to compare them. The main purpose of this first comparison is to get an overview on the functionalities of such kind of tools and to find out which meta-level objectives are already covered by tools.

In a first conclusion of this comparison we can say that the syntax definition as well as model transformation is already well captured by Segravis tools. These VL aspects are defined based on MOF and graph transformation. Interpreters and generators are available to provide the language designer 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 by tool support to such extent.

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

Meta CASE functionalities – general and syntax aspects: Segravis Tools

Name	AGG	DaMoRE	GenGED	CD-VML-UC	AToM3	Fujaba	MetaEnv	Viatra2	ConsistencyWorkbench
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
approach for high-level description	type graphs / graph transformation	graph transformation	typed attributed graph transformation	Meta modeling	Meta modeling for syntax, graph transformation for semantics	?	type graphs/graph transformation	VPM metamodeling, graph transformation, ASMs	graph transformation rules and activity-diagram like description for consistency checks
reference applications	Petri nets, statecharts, activity diagrams	bootstrap	Petri nets, Statecharts		Timed Automata, Process Interaction, DEVS, ...	MOF 2.0	PLCTools	model transformations for IBM Business Integrator	consistency checking of UML statecharts by translation into semantic domain CSP
special application domains	none	tool integration, reengineering			(multi-formalism) simulation	no	any	no	
syntax features	abstract syntax	any	concrete and abstract syntax	concrete/abstract syntax	concrete and abstract syntax	many	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	
allowed structures	typed, attributed nodes and edges	MOF package structure	by graph grammar	n/a	defined by meta-modelling	compatible with UML class and package diagrams	by an attributed type graph	class/object diagrams	
concrete layout	graph-like	graph-like	diagrammatic or icon-based		icon-based, no algorith for layout	graph-like	graph-like	tree view based	
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	Meta models + graph transformation rules + ASM programs	
interpreting editor	syntax-directed	no	syntax-directed		combination free hand / syntax directed	no		no	no
generated editor	none	yes	none		Yes	yes	none	no	no
parsing	For textual format	Java	For textual format		No	Java	none	for a textual format	
syntactical constraints	Graph constraints	OCL constraints		Constraints	Yes	Class diagram	none	metamodel inheritance, instantiation, (restricted) multiplicities	

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

Meta CASE functionalities – semantics aspects: Segravis Tools

Name	AGG	DaMoRE	GenGED	AToM3	Fujaba	MetaEnv	Viatra2	Consistency Workbench
operational semantics	yes	yes	yes	yes	yes	no	graph transformation + ASMs	for consistency checks
semantical domain		graph transformation		graph transformation, Python	graph transformation	Petri nets		
description	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 and interpreted
interpreting simulator	Visual, discrete, interactive/ automatic	no	Visual, discrete/ continuous, interactive/ automatic	Discrete/ continuous, interactive/ automatic	no	by means of token game	yes	for consistency checks
generated similator	No	no	for animation scenarios		no	none		
animation views	No	no	yes	yes	no	in the target editor	no	
animation features	No	no	continuous movement, color/size/ visibility changes	yes	Dynamic Object Browsing System (DOBS)		no	
Testing of model dynamics	Rule applications, no generation of test cases	JUnit	No	Rule applications	JUnit		no	
validation of model dynamics	conflicts, reachability of graphs	incremental analysis	No	simple model checker	no	benchmarks	no	

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