A Survey of Simulation Tools for Requirements Engineering

Final report of the work group "Simulationswerkzeuge für das Requirements Engineering". The work group was part of the Special Interest Group on Requirements Engineering (FG 2.1.6) of the German Informatics society (GI).

Reto Schmid, Johannes Ryser, Stefan Berner, Martin Glinz
Department of Information Technology, University of Zurich
{rschmid, ryser, berner, glinz}@ifi.unizh.ch

Ralf Reutemann
Daimler-Chrysler Aerospace AG
reutemann.ralf@dornier.dasa.de

Erwin Fahr
Berufsakademie Ravensburg
fahr@ba-ravensburg.de

Abstract

Validation of requirements specifications is undoubtly an integral and indispensable part of requirements engineering. Validation is the process of checking whether requirements specifications meet the intentions and expectations of the stakeholders. One approach to support the process of validation is based on simulation/execution and animation of system (behaviour) models that are derived from initial requirements specifications. However, the benefit of executable models is determined by the capabilities of the corresponding simulation tools. This paper presents a survey on simulation and animation capabilities of ten modern software/system engineering tools.

Table of Contents

A Survey of Simulation Tools for Requirements Engineering

1. Introduction

1.1 Motivation

Nowadays the role and importance of requirements engineering (RE) is widely undisputed. The later errors and misunderstandings are uncovered in a system development project, the more costs arise for fixing them [Boehm 81]. Thus, getting the requirements right the first time is a vital, although difficult, endeavour. Different terminologies and understandings of stakeholders and requirements engineers make good communication and therefore good RE specifications a hard task. Incompleteness, ambiguity, inconsistencies, and vagueness, to name just a few, are common problems encountered when eliciting and specifying requirements.

Due to this, validation of the requirements specification is an integral and indispensable part of RE. Validation is the process of checking, together with the stakeholders, whether the requirements specification meets the stakeholders' intentions and expectations [McDermid 94]. One approach to support the process of validation is based on simulation (execution) and animation (visualisation), provided that the requirements specification will be transformed into (semi-) formal models that can be executed. However, the benefit of executable models is determined by the corresponding simulation tools. Which tools exist and which simulation and animation capabilities do they provide? This document presents a survey of simulation and animation capabilities of 10 software/system engineering tools, evaluated from an RE point of view.

1.2 Process overview of the survey

In February 1997, we formed a work group, which was part of the Special Interest Group on Requirements Engineering (FG 2.1.6) of the German Informatics society (GI). The work group initially consisted of 3 members from industry and 5 members from academia. Our goal was to evaluate and compare simulation and animation capabilities of selected, modern RE/SE tools. For the sake of feasibility, we decided first to do a broad search of RE/SE tools with simulation/animation capabilities and then to conduct the survey on a few selected tools.

From February 1997 until May 1997, we worked on a preliminary questionnaire, which served as a basis for deciding whether a tool was a candidate for this survey or not. Until June 1998, we have been looking for commercial and research tools that fitted the scope of this survey (see chapter 3.1). The Internet, literature and hints from organisations like INCOSE were our main information sources. In parallel, we started gathering information about the tools we found so far, guided by the preliminary questionnaire. During the regular work group meetings, we decided which tools were candidates for the survey. From August 1997 until December 1997, we prepared a detailed questionnaire for evaluating the selected candidates. When a candidate had been chosen for further evaluation, we tried to obtain more information, based on the detailed questionnaire, and a demo version. Starting in November 1997, we specified a reference model of a fictive ticketing system. This model was to serve as a common basis for evaluating the tools. But implementing/ modelling this reference model in the various tools proved to be too time consuming. So we dropped this idea in May 1998. Instead, we decided to work with the tutorial models that are usually delivered together with the tools.

In June 1998, we stopped the (active) search for more tool candidates. By then, we had created a list of 54 tools, of which 16 were candidates for further evaluation according to our survey scope (see chapter 3.1). In order to have a reasonable amount of work, we selected 10 out of the 16 candidates for detailed evaluation, aiming at high diversity of modelling notations, geographical origin and distinguishing capabilities. Until January 2000, we conducted the evaluation of these 10 tools, based on demo versions whenever possible (see results in chapter 4). Overlapping with the evaluation, we began in June 1999 to write down the results of the survey in a technical report. We finished the survey in January 2000 and the technical report in May 2000.

1.3 Structure of the document

In chapter 2, we define the basic terminology used throughout this document, followed by a discussion of benefits and drawbacks of using simulation/animation for validating requirements. We define the tool capabilities according to which we evaluated the simulation tools.

In chapter 3, we outline the methodology we followed in the survey. We define the scope of the survey (criteria for tool candidates). Furthermore, we describe the survey process and the questionnaires we used for conducting the survey.

In chapter 4, we present the surveyed tools and briefly describe them. A comparison of the evaluated simulation and animation capabilities of each tool follows.

Chapter 5 lists the tools that we didn't evaluate. Finally, in chapter 6, we discuss the results and present the conclusions drawn from the survey's findings.

1.4 Acknowledgements

We would like to thank G. John, S. Galli and M. Arnold, who had to leave our work group before the survey ended, for their contributions.

Furthermore, we would like to thank Berner&Mattner Software Produkte GmbH (local distributor of I-Logix products), CIP SYSTEM AG, Excel Software, IBE Software und Simulation Engineering, Telelogic SA and Vitech Corporation, who provided demo versions of their products for us. Special thanks goes to CIP SYSTEM AG and Telelogic SA, who additionally gave us on-site tool demonstrations.

1.5 Disclaimer

We conducted this survey to our best knowledge and conscience. But the enormous number of tools that are related to requirements/software engineering makes it almost impossible to examine them all in depth. Additionally, simulation and animation capabilities strongly depend on each tool's modelling languages, which further complicates any comparison.

Due to the factthat this survey took almost 3 years, and due to the varying difficulty in obtaining the desired information, the survey results might be partially inaccurate today. The 'last update' information in the description of every tool indicates the accuracy of the information given.

2. Simulation/animation - why and what for?

For the sake of clarity, this chapter first lists definitions of the basic terminology used throughout this document. A discussion of benefits and drawbacks of simulation/animation in the context of validating requirements follows. Finally, we describe the tool capabilities, according to which we evaluated the simulation tools.

2.1 Definitions

System model - Formal or semiformal model of a computer-based system. Principally, a system model is an abstract, idealised and constructive solution of the problem that the system shall solve. In this document, we do not consider any declarative approaches of specifying computer-based-systems.

Model-based RE/SE - RE/SE methodology that is based on refining and validating an initial requirements specification by means of developing executable system models.

Modelling language - Language that is used to formulate system models. In the context of this survey, modelling languages are assumed to be at least semi-formal.

Notation - Graphical and/or textual representation of a modelling language.

Simulation - In the context of RE/SE, simulation is defined as execution of a system model. The modelling language used to express the system model must rely on defined execution semantics. Based on the semantics, a simulator tool can execute the model, either by direct interpretation or by code generation.

Animation - We define animation as the visualisation of the behaviour of the system model. Animation is often based on the (graphical) language/notation, in which the system model is expressed, e.g. by highlighting the current model element under execution. For better convenience, certain tools allow additionally the use of an application-specific graphical user interfaces (GUI).

Prototyping – Classical prototyping is based on partial implementations of the intended future system [Budde et al. 92], derived from the initial requirements specification. Depending on scope and feasibility several different prototypes might be needed. The implementation often includes an application-specific GUI as primary way of interacting with the prototype. As with simulation, prototyping can also be model-based. In this case, execution is achieved by generating instrumented code.

Validation - The process of evaluating software during, or at the end of, the development process to determine whether it satisfies the specified requirements [IEEE 90]. The process of questioning whether the right product is being built.

Verification - (1) The process of evaluating software to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase; (2) formal proof of correctness [IEEE 90]. The process of questioning whether the product is being built correctly.

Architectural and detail level - Some modelling languages focus on the architecture (composition) of a system, whereas execution "details" like object methods have to be formulated in a different (usually standard programming) language. In this case, the architectural level corresponds to the modelling language, whereas the detail or code level encompasses the model details formulated in a programming language. See ROOM [Selic et al. 92] as an example which incorporates such a modelling concept.

2.2 Benefits of simulation/animation

Validation of a concrete object - As with prototyping, simulation has the major benefit of validating a concrete and executable object (the system model), which can improve the quality of a requirements specification. You are not only bound to inspections and reviews of textual requirements specifications.

No need to write code - In contrast to classical prototyping, principally no code must be (manually) written in order to run simulations of system models. Depending on the simulation tool used, a model may require code fragments in order to implement specific execution details.

No need to create application-specific GUIs - Most simulation tools directly animate the graphical notation of the supported modelling language(s). As long as all parties working with simulation/animation are familiar with the animated modelling languages, there is no (immediate) need for creating application-specific GUIs in order to interact with the executed models.

White-box visualisation - Animation of modelling languages also provides a means of visualising the internal mechanisms (behaviour) of the system model. That way, one has the possibility of validating the internal mechanisms of a system model by inspection. In contrast, classical prototyping usually relies only on application-specific GUIs for interacting with the prototype (black-box view).

Better understanding of the system - Simulation/Animation helps to better understand the (desired) behaviour of the future system. Quite often the stakeholders are not sure about what they exactly want or how to voice their ideas and desires. Additionally, a well-understood system allows to predict development costs more accurately.

2.3 Drawbacks of simulation/animation

No way to validate discrete systems exhaustively - Simulation/animation shares the major limitations and drawbacks of testing. Complex systems with a large number of states ("combinatorial explosion") can neither be tested nor simulated exhaustively in general. Furthermore, repeating validation activities after every change made (regression testing) - a good engineering practice - adds even more work.

(Standard) Model animation is not always stakeholder-friendly - The standard way of model animation is based on highlighting the current state of the model, which in turn is represented by the notations of the supported modelling languages. Non-technical stakeholders, who are no familiar with RE/SE languages, might experience difficulties in reading and interpreting such animations.

Low abstraction level - Simulation and prototyping often require additional implementation details of the future system in order to be executable at all. This leads to the problem of over-specification (too much solution/implementation-oriented).

Semantically correct and formal models required - As long as the RE process is in progress, the requirements engineers have to deal with incomplete and informally stated requirements. Nevertheless, in that case requirements engineers are forced to define a semantically correct and formal system model in order to run simulations.

Need to model the system's environment - Embedded systems form a part of larger systems and involve complex interaction with their environment. A reasonable simulation of an embedded system also requires to model and simulate the system's context.

Non-functional requirements not fully supported - Simulation focuses on the behavioural aspects of a system. However, non-functional requirements, such as reliability, cannot be simulated adequately from requirements.

2.4 Tool capabilities

This chapter presents the tool capabilities that we have taken into account for conducting this survey. The tool descriptions and comparison in chapter 4 are based upon these capabilities.

2.4.1 Execution

There are two basic ways of executing a model:

Interpreter based- The simulator directly interprets the system model. There is no need for an external compiler, but interpretation is more likely to be slower than its compiled counterpart. Nevertheless, this is of no major concern for a lot of (RE) problem domains. Interpretation doesn't directly support an automatic shift to subsequent development phases by simply generating the code from the system model.

Code generation - The tool set translates the model into a conventional programming language like Java or C++, then calls an external compiler to produce the executable. The code is instrumented in order to interact with the simulator during the simulation run. This way, simulation runs are still controlled and manipulated via the simulator tool. The use of an external compiler can be well encapsulated within the simulator tool, so the tool user won't notice any relevant difference to direct interpretation. Code generation does not guarantee that the generated code can be seamlessly reused in the following design and implementation phases, since the code might have a complex and unreadable structure. Generated code often has to be used as is.

Some problem domains (e.g. embedded systems) require cross-platform development already during the RE phases:

Cross-platform execution - Interpreted system models run on platforms supported by the simulator tool itself or by a virtual machine. In contrast, generated code often can be compiled for quite a big number of platforms, due to the high availability of cross-compilers.

2.4.2 Interactive control of simulation

We have evaluated the following forms of user interaction with a simulation tool:

Step forward - In order to control the simulation progress interactively, the tool should provide controls similar to those known from video cassette recorders (VCR): go single step, go until idle, go until next event, go infinite, pause, reset, etc.

Backward steps (undo function) - Sometimes, a tool user even needs go back one step, go back to time T, or the like.

Generate events - The simulation tool should provide some appropriate means to generate events (stimuli) that originate from the system environment. This could be accomplished by selecting the desired language elements (e.g. a state transition) and invoking the corresponding command via buttons or menus (e.g. fire the transition). The simulation tool could also display a list of currently accepted stimuli, from where the tool user selects the one to be executed next.

Modify model state - Some tools allow to directly modify the current state of the system model, e.g. by assigning new values to variables.

2.4.3 Automated control of simulation

Besides working interactively with a simulation tool, a tool user often needs an automated control of simulation run. E.g., automated control simplifies regression testing to a great extent. We have taken the following two forms of automated control into account:

User-defined scripts - A simulation tool should be able, according to our opinion, to run user-defined (test) scripts in order to control simulation runs. TTCN (ITU Z.120) and MSC are examples of languages that are suited for writing simulation scripts.

Replay of previous runs - Furthermore, a simulation tool should support the replay of previously recorded simulation runs. E.g., MSCs can serve as a format for storing simulation runs.

Parametrisable stimuli generators are a further way of automatically controlling simulation runs [Bloss et al. 92]. But we expect such generators to be highly problem domain depended. Therefore, none of the tools, we surveyed, provided any stimuli generator.

2.4.4 Animation

Simulation is closely related to animation, the visualisation of the behaviour of the system model. We have evaluated the simulation tools according to the following capabilities:

Graphical notation(s) - Depending on the modelling languages supported, the simulation tool either animates the graphical or the textual notation(s). Languages like Statecharts, DFD, SDL, Petri Nets, etc. provide a natural and intuitive way of animation, by highlighting the current focus (i.e. the currently executed model element). The expressiveness of an animation therefore depends largely on the modelling languages supported by a tool.

Customisable Graphical User Interface (GUI) - When working with stakeholders who are not familiar with RE/SE specification languages, the requirements engineer needs to specify application-specific and tailored views (GUIs). Such GUIs simplify the interaction between stakeholders and system models. If the simulation tool directly supports the definition and application of such GUIs, we name it “customisable GUI”.

External GUI - If the simulation tool does not provide customisable GUIs, you could alternatively implement the GUI with a separate development system, hence we call it external GUI. Typically, tools that are based on code generation support external GUIs, since the GUI code can be linked to the generated code of the system model.

Animated real-time - The simulator can animate the system model by constant speed, taking timing constraints into account only for correctly scheduling events/stimuli (simulated real-time). Alternatively, the animation can also be based on real-time constraints, resulting in an animated "real-time". In both situations the animation speed should be configurable.

Animation of code fragments - Apart from the architectural level, some tools additionally require that execution "details" (e.g. actions of a state transition) are formulated in a standard programming language. This gives rise to the question, whether or not a simulation tool animates such code fragments too.

Generally the simulator user should be able to select the views of interest, e.g. selecting some of the model components. Support for navigating and browsing through a model is also needed. Being able to put the focus of interest only on selected parts of the model is very important, in order not to drown in a sea of details. See for example approaches like "fish eye views" [Berner et al. 98].

2.4.5 Recording simulations

Animations of system models typically visualise the current state of the models. Tool users often need condensed information recorded over a defined period of time during a simulation run.

Traces/logs - The simulator logs all or a defined subset of the events that occur during a simulation run. E.g., the communication among several components of a system model may be recorded as a MSC.

Statistics - subsumes any kind of statistical information about elements of the system model. Average duration of states, average response times and maximum input queue length are examples of such statistics. Statistical data can be graphically displayed e.g. as histogram, pie chart or bar chart.

Time lines - denote the visualisation of the state of a system model by plotting a 2D graph. The horizontal X-axis corresponds to the time, whereas the vertical Y-axis is a nominal scale of the states which may be observed during recording. Time lines are similar to oscilloscopes.

2.4.6 Validation

Validation can hardly be (fully) automated. But a tool set should at least support the management and conduction of validation activities. We have chosen the following two capabilities for evaluating the simulation tools:

Model coverage views - The tool computes and visualises the coverage of a system model during a single simulation run.

Validation management - We subsume any kind of validation support (e.g. model coverage) that reaches beyond one single simulation run under the term validation management.

2.4.7 Verification

Besides any validation capability, the enclosing RE/SE tool set (containing the simulator) should also support verification techniques. We have evaluated the following verification capabilities:

Syntax - The tool checks, whether the system model conforms to the syntactical rules of the supported modelling languages.

Statically checked properties - denotes any kind of (semantical) checksthat are performed before executing a system model. The tool user may even program further static checks or customise existing ones. Deadlocks, state reachability and theorem proving are examples of such a kind of verification.

Dynamically checked properties - denotes any kind of (semantical) checks that are performed during execution of a system model. The tool user may even program further dynamical checks or customise existing ones. Checking whether the system model respects the life cycle constraints of some object, may serve as an example.

Invariants - During simulation runs, a tool checks after each atomic execution step, whether the simulated system model does not violate user-defined invariants.

Compare scenario against model - A tool compares a recorded or manually written scenario (e.g. a MSC) against a system model.

State space exploration techniques - State space explorations are automatic simulation runs that do not require any intermediate interaction with the tool user. Such automatic explorations can be used to check dynamical properties of a system model. State space exploration techniques especially help to find run-time errors like buffer overflowsthat may be hard to detect with regular (i.e. user-controlled) simulation runs. We distingiush the following two forms of state space exploration techniques:

Random exploration - This state space exploration technique proceeds by selecting the system states randomly.

Limited depth exploration - This state space exploration technique proceeds by systematically selecting all consecutive system states up to a specified maximal depth.

2.4.8 Modelling languages

Table 1 shows characteristics and a categorisation of the modelling/specification languages that are used by the tools covered in this survey (see chapter 4.11.2). The categorisation (behaviour, communication/interaction, structure, others) of the languages is roughly based on [Wieringa 98]. We considered the following 4 groups of language characteristics:

Notation type - The notation of a modelling language can be graphical (with textual labels and annotations) or textual. Some languages, e.g. SDL, support even both notation types.

Abstractions - The most basic abstractions used in RE/SE are aggregation (hierarchical decomposition relation), inheritance (generalisation - specialisation relation) and delegation (client - server relation). Delegation specifies which services are requested by one system component from some other system component. Exemplary languages (e.g. MSC) rather define one specific scenario of a system, than specifying its complete behaviour.

Formality of semantics - Modelling languages have formally or semi-formally defined (execution) semantics. We consider languages to have formally defined semantics, if rigorous mathematical specifications of the language semantics do exist. Semi-formal languages typically have a formally defined syntax and grammar, whereas the semantics are defined in prose. Since execution of informal (i.e. natural) languages is still an extremely difficult (research) topic, they are rarely used for simulation. Therefore, we do not consider informal languages in this document.

	Notation type		Abstractions				Formality of semantics	
Modelling Languages	graphical	textual	aggregation	inheritance	delegation	exemplary	formal	semi-formal
Behaviour								
FSM	X	-	-	-	-	-	X	-
Statechart	X	-	X	-	(X)	-	X	-
Classical Petri Net	X	-	-	-	-	-	X	-
Coloured Petri Net	X	-	-	-	-	-	X	-
Hierarchical Petri Net	X	-	X	-	(X)	-	X	-
Pr/T Net	X	-	-	-	-	-	X	-
SDL Process diagram	X	-	-	(X)	(X)	-	-	X
Activity / control flow diagram	X	-	-	-	(X)	-	-	X
Communication/Interaction								
DFD	X	-	X	-	(X)	-	-	X
SDL Block diagram	X	-	X	(X)	(X)	-	-	X
UML use case diagram	X	-	(X)	-	X	-	-	X
UML interaction diagram / MSC 96	X	-	-	-	X	X	-	X
Context / interface diagram	X	-	-	-	X	-	-	X
Structure								
Class diagram	X	-	X	X	-	-	-	X
Object diagram	X	-	X	-	-	X	-	X
CRC card	-	X	-	X	X	-	-	X
Others								
Code fragments	-	X	(X)	(X)	(X)	-	(X)	(X)
Flow chart	X	-	X	-	(X)	-	-	X

Table 1 : Characteristics and Categories of Modelling Languages

The following symbols are used in Table 1:

X : The modelling language L has characteristic C.
(X) : The modelling language L has characteristic C in a restricted form.
- : The characteristic C does not apply to the modelling language L.

3. Methodology of the survey

In this chapter, we describe the methodology of our survey. In section 3.1, we present the scope of the survey by defining the capabilities a tool had to match in order to be chosen as a candidate. We also define the kinds of tools that we considered to be out of the survey scope. In section 3.2, we describe the process we followed during the survey. Finally, section 3.3 introduces our primary instruments, i.e. the two questionnaires used for gathering and structuring the information about the tools.

The aim of this survey was to find simulation toolsthat are suited for RE/SE, and to evaluate their simulation and animation capabilities. For the sake of feasibility we decided first to do a broad search of RE/SE tools with simulation/animation capabilities, then to conduct a survey on a few selected tools.

3.1 Scope of the survey

3.1.1 Tools to be considered for the survey

The number of tools to be evaluated must be of reasonable size. This section lists the criteria a tool had to match in order to be considered for the survey.

- Any tool that can be used for RE purposes in a broad sense was to be considered.
- According to the aim of this survey, each selected tool must provide some form of interactive simulation (execution) and animation (visualisation) of a system model.
- Related to simulation, but still substantially different, is prototyping. Since both techniques can be used for modelling and validating requirements, we had to decide to which extent to include prototyping tools in this survey. Model-based prototyping is quite similar to simulation. In this sense, as long as a prototyping tool is model-based and equipped with (hidden) code generation and animation capabilities, we considered it as a potential candidate for the survey.

3.1.2 Tools not to be considered for the survey

Any tool that belonged to one of the following categories was not considered:

- User interface prototypers
- Workflow / business process reengineering (BPR) tools
- General purpose (discrete event) simulation tools
- 4th Generation Language (4GL) CASE tools
- Reengineering tools
- Graphical programming tools

3.1.3 Functionalities not to be considered

Many of the tools we encountered during the broad search cover an impressive range of functionalities, which we decided to be outside of the survey scope. The following functionalities, although very important for real RE endeavours, were not taken into account for the survey:

- project management
- multi-user management
- quality assurance
- configuration/change management
- version control
- requirements identification
- requirements traceability

3.2 Process of the survey

Our work group conducted the survey according to the project schedule described in the following sections.

3.2.1 Founding a work group (Phase 0)

During phase 0, we formed a work group that conducted this survey. The work group was part of the Special Interest Group on Requirements Engineering (FG 2.1.6) of the German Informatics society (GI). As a first step, we defined the survey goals and its scope. The initial members were asked to acquire new members. A group size of 6 to 8 members was found to suit ideally the needs of this survey. At the end of phase 0, the work group consisted of 3 members from industry and 5 members from academia. All members are professionals in information science and have a special interest in RE.

Phase 0 started in January 1997 and lasted until February 1997.

3.2.2 Searching for tools (Phase I)

In the next phase we collected candidate tools. This phase was characterised by the following activities:

- We defined the categories of tools to be considered and those not to be considered.
- We gathered information about potential tool candidates:
  - by searching the internet
  - by asking organisations (e.g. INCOSE) about existing tools
  - by obtaining and studying technical papers about the tools
- We met as a work group on a regular basis in order to decide upon candidate tools, to check the status of the work and assign (new) tasks and responsibilities as necessary.
- We prepared a preliminary questionnaire that guided us in structuring information about the tools we considered for our survey. The questionnaire also served us to decide and chose tools as candidates for further evaluation.
- We prepared a detailed questionnaire for structuring information about the tools we evaluated in more detail.
- We specified a reference model which was to serve as a common basis for evaluating the tools. The reference model specified a fictive ticketing system. But implementing/modelling this reference model in the various tools proved to be too time consuming. So we dropped this idea in May 1998. Instead, we decided to work with the tutorial models that are typically delivered along with the tools. Since we didn't make use of our reference model, we decided not to describe it further in this document.

Phase 1 lasted from February 1997 until the end of 1998. But since the work of phase 1 partially overlapped with the subsequent phase 2, no strict border can be drawn. We ended phase 1 with a list of 54 tools, 16 of which met the criteria as presented in section 3.1. These 16 tools were chosen as candidates for the survey.

3.2.3 Evaluating the selected tools (Phase II)

Based on the information gathered, we started to evaluate 10 of the 16 candidates in more depth during phase 2, mainly guided by the detailed questionnaire (see appendix B). We aimed at a high diversity of modelling notations (UML, SDL, extended FSM, Statechart, data flow, Petri Nets and CRC), geographical origin (USA, Canada, GB, Germany, France, Sweden and Switzerland), popularity and distinguishing capabilities as criteria for selecting 10 tools out of the 16 candidates. For most of the selected candidates we could obtain a demo version to get some hands-on experience. The work group meetings were continued in order to check the status of the work and to discuss intermediate results.

Phase 2 started in June 1998 and ended in January 2000, when the closing work steps began.

3.2.4 Documenting the survey (Phase III)

Finally, having evaluated the selected tools, we wrote this technical report that summarises all information gathered, the experiences made and the results achieved by our work group. We concluded our work in May 2000.

3.3 Primary instruments of the survey

In order to co-ordinate and structure the information that we collected, we prepared two questionnaires. The preliminary questionnaire served to identify potential candidates from the list of tools found during this survey. The detailed questionnaire contains all information needed for the evaluation and comparison of the selected candidates.

3.3.1 Preliminary questionnaire

The preliminary questionnaire was used whenever we discovered new tools. The questionnaire focuses on the most important aspects of a tool, e.g. its simulation features. Guided by the preliminary questionnaire, we decided whether to continue with further evaluation of some tool or to place it on the "negative list". The preliminary questionnaire is reprinted in appendix A.

3.3.2 Detailed questionnaire

Based on the preliminary questionnaire, we developed an extended version. This detailed questionnaire was used for structuring and storing information gathered about the tools that were evaluated in more detail. The detailed questionnaire is reprinted in appendix B.

4. The surveyed tools

Out of the 54 tools initially considered, we selected the following 10 tools for a detailed evaluation and comparison:

- CIP Tool 3.0
- COREsim 2.1
- ObjectGEODE 4
- ObjecTime Developer 5.0
- PACE 3.1
- QuickCRC 1.2
- RDD-100 / DVF 4.1
- Rhapsody for C++ 2.1
- SDT / ORCA 3.3
- Statemate MAGNUM 1.3

All 10 tools are equipped with simulators according to section 3.1. We had 6 other tools on our candidate list (see 5.4) that satisfy the survey scope as described in section 3.1, but for the sake of a reasonable amount of work we focused on those 10 tools that are presented in this section.

In sections 4.1 to 4.10, each of the 10 selected tools is shortly described. The description of the tools is followed by a comparison in section 4.11. Please see chapter 7 for references and sources of materials. The tool capabilities that we refer to in the following descriptions have been defined in chapter 2.4.

We assumethat the reader is familiar with nowadays languages used for requirements and software engineering, languages like Specification and Design Language (SDL), Unified Modeling Language (UML), Statecharts, Finite State Machine (FSM), Class Responsibility and Collaboration cards (CRC cards), Petri Nets and Data Flow Diagram (DFD).

4.1 CIP Tool 3.0

Vendor:

CIP SYSTEM AG, Switzerland

Tool set:

CIP Tool 3.0 is a commercial standalone product.

Supported Languages:

- Communication Net; *a kind of data flow diagram between clusters of synchronous processes*
- Interaction Net; *a kind of dataflow diagram within a cluster*
- Cascades; *tree-like graphs depicting the control flow between processes of one cluster*
- State-Transition-Diagram *(for process modes); an FSM*
- Master-Slave Graph; *directed graph-like control flow between process modes*
- C code *for (Boolean) transition guards*

Simulation/Animation Capabilities:

Execution: Instrumented C code has to be generated before the simulation/animation can take place.
Control: CIP Tool provides the basic controls: single step and reset. During a simulation run you can generate/inject stimuli. When working with textual animation, you can optionally load and execute a simulation log. Such simulation logs can also be manually written.
Animation: Textual as well as graphical animation is provided. The textual animation (a kind of debugger) is mainly intended for cross-platform debugging. During simulation runs the current states are highlighted.
Recording: Simulation runs can be recorded as a text file, listing the events (messages) and the time of their occurrence.
Verification: Verification of the model's correctness (syntax and semantics) is provided.

Distinguishing Features:

- Modelling of multiplicity (of object instances)
- Interaction Graph; abstract behaviour visualisation
- The tool originates from a research project. Its modelling languages are based on very solid formal basis, but do not conform to any widely known standards.

Information Sources:

- demonstration of the tool
- demo version
- web pages
- last updated 3.12.1999

4.2 COREsim 2.1

Vendor:

Vitech Corporation, USA

Tool set:

COREsim 2.1 is part of the commercial CORE 2.1 tool set.

Supported Languages:

- Element Relationship Attribute (ERA); *Entity-Relationship-Model of requirements*
- Function Flow Block Diagram (FFBD); *a kind of data flow diagram*
- Enhanced FFBD (EFFBD); *annotated Function Flow Block Diagram*
- Interface chart (N2); *a kind of context diagram*

Simulation/Animation Capabilities:

Execution: The system model is directly interpreted.
Control: COREsim provides the basic controls: run, single step, pause and reset. During a simulation run one can edit the system model, but you can't change the model state. Events are automatically selected (no human interaction necessary), .i.e. that any process (function block) may run provided that the required resources (another model element type) are available.
Animation: During simulation, FFBD/EFFBD diagrams highlight the current function/activity under execution. Additionally a timeline for various model elements can be displayed.
Recording: Simulation transcripts (logs) can be enabled and saved anytime. Transcripts are simple text files listing all events and the time of their occurrence.
Validation: Model coverage can be (manually) deduced from timelines.
Verification: Verification of the model's correctness (syntax and semantics) is provided.

Distinguishing Features:

- Extensive, all-purpose system engineering tool set
- Complex and adaptable database schema.
- The database encompasses requirements management and system modelling in parallel
- Extensive scripting language for reporting

Information Sources:

- demo version
- manuals (paper)
- web pages
- last update 20.12.1999

4.3 ObjectGEODE 4

Vendor:

VERILOG S.A., France

Tool set:

ObjectGEODE 4 is a commercial standalone product.

Supported Languages:

- SDL
- MSC
- UML 1.x

Simulation/Animation Capabilities:

Execution: ObjectGEODE supports both direct interpretation of models and code generation.
Control: ObjectGEODE provides the typical controls: various forms of (single) step, run, pause and reset. Additionally, backward stepping is supported. During simulations one can inject messages (stimuli). MSCs, either manually written or recorded from previous simulations, can be used for driving simulation runs.
Animation: Animation is based on highlighting the current state in the various SDL diagrams.
Recording: Simulation runs can be recorded as MSCs.
Validation: For each single simulation run a coverage view can be displayed.
Verification: Verification of the model's correctness (syntax and semantics) is provided. Besides, ObjectGEODE provides automatic state space (random, exhaustive/limited depth) exploration. Invariants can be formulated and attached to models, which are checked during simulation runs. Observers can be programmed and set-up for checking dynamic model properties during simulation runs. MSCs can be checked against the model.

Distinguishing Features:

- Wide range of verification techniques (state space exploration, invariants, observers).

Information Sources:

- web pages
- last update 30.10.1999

4.4 ObjecTime Developer 5.0

Vendor:

ObjecTime, Canada

Tool set:

ObjecTime Developer 5.0 is a commercial standalone product.

Supported Languages:

- Real -Time Object Oriented Modelling (ROOM) (architectural level); *Statechart-like language*
- UML 1.x
- C++ (code level)

Simulation/Animation Capabilities:

Execution: The Developer simulator (Simulation RTS) is interpreter-based. Optional modules allow code generation for C++ for various platforms. Generated code can be linked with MicroRTS in order to connect a model on a different (target) platform to Developer.
Control: Developer provides the typical simulation controls: go step, go event, go until idle, go continuously. During a simulation run one can manipulate the system model: inject events/messages and modify model attributes.
Animation: During simulation runs the current state is highlighted in the ROOM-models.
Recording: You can invoke a simulation event tracethat can be converted into a MSC.
Verification: Verification of the model's correctness (syntax and semantics) is provided.

Distinguishing Features:

- Controlling and debugging the model on a remote target platform (MicroRTS).
- Converting event traces into MSCs.

Information Sources:

- web pages
- literature [Selic et al. 92]
- last update 30.10.1999

4.5 PACE 3.1

Vendor:

IBE Software und Simulation Engineering, Germany

Tool set:

PACE 3.1 is a commercial standalone product.

Supported Languages:

- Hierarchical Petri Net (architectural level)
- Smalltalk (code level)

Simulation/Animation Capabilities:

Execution: The Petri Net models are directly interpreted by PACE. Optionally PACE can generate specialised and proprietary N-code for a separately available Petri Net Machine (PNM), a kind of virtual machine for PACE Petri Nets. This virtual machine is available for various platforms.

Control: PACE provides the typical simulation controls: various forms of (single) step, run until event X occurs, run until idle and reset. Additionally one can step backwards. During a simulation run you can manipulate the system model by injecting or deleting tokens, setting variable values and modifying the model. The model itself can also be edited when a simulation run is paused.

Animation: The flow of tokens is displayed. The window that contains the current token is automatically brought to the front. The animation can be individually enabled for each window that displays a part of the system model. The speed of token animation can be defined within given limits. The simulated "real-time" is taken into account for scheduling of the events, but not for animation. Simple customised GUIs can be built from a limited set of GUI elements.

Recording: One can save the current model state any time and reload it later. Ready-to-use means for collecting and displaying statistical data (histograms) is provided.

Verification: Verification of the model's correctness (syntax and semantics) is provided. Additionally, PACE checks for unused model elements.

Distinguishing Features:

- Ready-to-use GUI elements for interacting with the system model as well as for displaying statistical data.
- Since the various model elements (places, transitions, tokens) can be "inscribed" with Smalltalk code the PACE Petri Nets can be programmed towards different net types like Predicate/Transition (Pr/T) -Nets or Coloured Petri Nets (CPN).

Information Sources:

- demo version
- manuals (online and paper)
- last update 18.06.1999

4.6 QuickCRC 1.2

Vendor:

Excel Software, USA

Tool set:

QuickCRC is a commercial standalone product.

Supported Languages:

- Class, Responsibility and Collaboration (CRC) cards
- Scenario card; *sequence of client object - sever object - responsibility or sub-scenario calls*
- Class inheritance graph

Simulation/Animation Capabilities:

Execution: QuickCRC is interpreter-based.
Control: The simulator provides the basic controls: single step forward, single step backward, reset, go to end, step over, back to caller.
Animation: The simulation corresponds to a textually displayed scenario walkthrough. The walkthrough is displayed in a single window containing the following elements:

- list containing scenario and its sub-scenarios (indenting shows the call hierarchy)
- description of the current active scenario
- textual formulation of the current step in a format like “Client object X uses Responsibility / Scenario Z of Server Object Y”.

Distinguishing Features:

- QuickCRC is a tool setthat supports the Responsibility Driven Design (RDD) development method [Wirfs-Brock et al. 90].

Information sources:

- demo version
- web pages
- last update : 26.09.1999

4.7 RDD-100 / DVF 4.1

Vendor:

Ascent Logic Corporation, USA

Tool set:

RDD-100 / DVF 4.1 is part of the commercial RDD-100 tool set.
RDD-100 has been replaced by RDD.COM in the mean time.

Supported Languages:

- Stimulus Response diagram; *mixture between FSM and SDL process diagram*
- Behaviour graph
- Interconnection
- Operational scenario

Simulation/Animation Capabilities:

Execution: RDD-100 / DVF is interpreter-based.
Control: DVF provides the typical simulation controls: go step, go continuously, pause, restart. During a simulation run one can manipulate the system model by injecting events/messages and modifying model attributes.
Animation: During a simulation run, various views are animated: behaviour diagrams (highlighting the current state), time-lines, resource graphs (resource availability), and queue (usage) monitors.
Recording: Simulation runs can be recorded as event traces (transcripts).
Verification: Verification of the model's correctness (syntax and semantics) is provided.

Distinguishing Features:

- Extensive, all-purpose system engineering tool set

Information Sources:

- web pages
- marketing brochures
- last update 21.10.99

4.8 Rhapsody for C++ 2.1

Vendor:

I-Logix Inc., USA

Tool set:

Rhapsody for C++ 2.1 is a commercial standalone product.

Supported Languages:

Rhapsody is UML 1.2 based

- Object model diagram (i.e. class/object diagram)
- Statechart
- Sequence diagram
- Use case diagram
- C++ code fragment

Simulation/Animation Capabilities:

Execution: Prior to simulation the system model must be translated to code (C / C++) and then compiled by an extra compiler (like VC++ from Microsoft). The code is instrumented to either interact with the graphical simulator of Rhapsody or with Tracer (a kind of text-based debugger).

Control: Rhapsody provides the typical simulation controls: go step, go event, go until idle, go continuously. During a simulation run you can manipulate the system model: inject events and modify model attributes.

Animation: During simulation, Statecharts, sequence diagrams, model browser, thread view, event queue and call stack are animated. The simulated "real-time" is taken into account for scheduling events and optionally also for animation. Code fragments are animated as a single, atomic action. The Tracer provides a textual simulator/animator.

Recording: Simulation runs can be recorded as sequence diagrams.

Verification: Verification of the model's correctness (syntax and semantics) is provided. Additionally, Rhapsody offers various checks like detecting unused model elements. Sequence diagrams (recorded or manually written) can be compared two by two.

Distinguishing Features:

- Round trip engineering
- Interoperability with version control tools
- Interoperability with requirements management tools
- Comparison of sequence diagrams

Information Sources:

- demo version
- manuals (online)
- web pages
- marketing brochures
- last update 22.10.1999

4.9 SDT / ORCA 3.3

Vendor:

Telelogic SA, Sweden

Tool set:

SDT / ORCA is part of the commercial Telelogic Tau 3.3 tool set.

Supported Languages:

- SDL '92 and SDL '96
- MSC '96
- HMSC; *hierarchical variant of MSCs, which are composed from HMSCs and MSCs*
- Class diagram (UML)
- Statechart (UML)
- ASN.1
- Proprietary notation for abstract data type definition; *alternative to ASN.1*

Simulation/Animation Capabilities:

Execution: Prior to simulation the system model must be translated to code (C / C++) and then compiled by an extra compiler (like VC++ from Microsoft). The code is instrumented to interact with the SDT Simulator.

Control: SDL provides the typical simulator controls: start, restart, execute next transition(s), execute next statement(s), go until time == T, go until some event occurs, go until idle, etc. Additionally you can step backwards. During a simulation run you can manipulate the system model via monitor: injecting signals, setting variable values, creating or terminating processes, setting or resetting timers, setting the state of a process and more.

Animation: The animation is based on highlighting the current SDL statement, highlighting the current state of a Statechart and tracing the signals exchanged between model components as an MSC. The simulated "real-time" is taken into account for scheduling the events, but not for animation. The animation proceeds as fast as possible.

Recording: Simulation runs can be recorded as MSCs and can be replayed afterwards.

Validation: The model coverage during simulation is measured.

Verification: Verification of the model's correctness (syntax and semantics) is provided. Besides, SDL/ORCA checks also for unused model elements. Additionally, you can check MSCs against the modelled behaviour.

Distinguishing Features:

- Validator that conducts state space exploration (manual, random walk, limited depth)
- Verify MSCs against the system model (SDL)
- Model coverage view
- An API is provided so that external applications can interact with the simulated system model.

Information Sources:

- demonstration of the tool
- demo version
- manuals (paper)
- web pages
- last update 16.04.1999

4.10 Statemate MAGNUM 1.3

Vendor:

I-Logix Inc., USA

Tool set:

Statemate MAGNUM 1.3 is a commercial standalone product.

Supported Languages:

- Module chart
- Activity chart
- Statechart

Simulation/Animation Capabilities:

Execution: The simulator (Trailblazer) is interpreter-based. Optional modules allow code generation for various languages.
Control: The simulator provides the typical simulation controls: go single step, go event (next stable state), go extended (next stable superstate), go advance (simulation time + T), go repeat. During a simulation run you can manipulate the system model: injecting events/messages and modifying model attributes. A script language (SCL) allows to automate simulations runs.
Animation: During simulation views of activity charts and Statecharts are animated. The simulation is animated in "real-time".
Recording: Simulation runs can be recorded as event traces.
Validation: During simulation a (test) coverage is measured and can be viewed afterwards.
Verification: Verification of the model's correctness (syntax and semantics) is provided.

Distinguishing Features:

- limited, built-in version control
- simulation control language (SCL)
- requirements tracing
- coverage measure
- interface to Requirements Management Systems
- reporting on simulation runs
- API for model access
- API for custom code generation
- API for requirements tracing

Information Sources:

- access to a full installation at ETHZ (Swiss Federal Institute of Technology Zurich), Switzerland
- manuals (online)
- web pages
- last update 15.10.1999

4.11 Comparison of the tools

4.11.1 Definition of the symbols used

The following symbols are used in the tables of section 4.11.2 and 4.11.3. The symbols form a nominal scale.

X	:	Tool T fully supports capability C/language L.
X+	:	Tool T fully supports and extends capability C/language L.
(X)	:	Tool T supports capability C/language L in restricted form.
-	:	Tool T does not (explicitly) support capability C/languageL.
n/a	:	Comparison between tool T and capability C/language L is not applicable.

4.11.2 Supported languages

Based on section 2.4.8, Table 2 shows which tool supports which languages/notations.

	CIP Tool 3.0	COREsim 2.1	ObjecTime Developer 5.0	ObjectGEODE 4	PACE 3.1	QuickCRC 1.2	RDD-100 / DVF 4.1	Rhapsody for C++ 2.1	SDT / ORCA 3.3	Statemate MAGNUM 1.3
Behavioural										
FSM	X+	-	-	-	-	-	(X)	-	-	-
Statechart	-	-	X+	X	-	-	-	X	X	X
classical Petri Net	-	-	-	-	X	-	-	-	-	-
Coloured Petri Net	-	-	-	-	(X)	-	-	-	-	-
hierarchical Petri Net	-	-	-	-	X	-	-	-	-	-
Pr/T Net	-	-	-	-	(X)	-	-	-	-	-
SDL Process diagram	-	-	-	X	-	-	(X)	-	X	-
Activity / control flow diagram	X	-	-	-	-	-	-	-	-	-
Communication / Interaction										
DFD	(X)	(X)	(X)	-	-	-	(X)	-	-	X+
SDL Block diagram	-	-	-	X	-	-	-	-	X	-
UML use case diagram	-	-	-	-	-	-	-	X	-	-
UML interaction diag./ MSC '96	-	-	X	X	-	-	(X)	X	X+	-
Context / interface diagram	-	X	-	-	-	-	-	-	-	-
Structural										
class diagram	-	-	-	X	-	(X)	-	X	X	-
object diagram	-	-	-	-	-	-	-	(X)		-
CRC card	-	-	-	-	-	X	-	-	-	-
Others										
Mandatory code fragments	-	-	X	-	-	-	-	X	-	-
Optional code fragments	(X)	-	-	-	X	-	-	X	-	-
Flow chart	-	(X)	-	-	-	-	(X)	-	-	-

Table 2 : Languages Supported by the Surveyed Tools

4.11.3 Simulation/animation capabilities

Tables 3 and 4 summarise the capabilities of the surveyed tools, based on the brief descriptions of the tools in sections 4.1 to 4.10. A discussion of the capabilities can be found in section 2.4. Please see section 4.11.1 for a definition of the symbols used in the table.

	CIP Tool 3.0	COREsim 2.1	ObjecTime Developer 5.0	ObjectGEODE 4	PACE 3.1	QuickCRC 1.2	RDD-100 / DVF 4.1	Rhapsody for C++ 2.1	SDT / ORCA 3.3	Statemate MAGNUM 1.3
Execution										
Interpreter based	-	X	X	X	X	X	X	-	-	X
Code generation	X	-	X	X	-	-	-	X	X	X
Cross-platform execution	X	-	X	X	X	-	-	X	X	X
Interactive Control of Simulation										
Step forward	(X)	X	X	X	X	(X)	X	X	X	X
Step backwards (undo function)	-	-	-	X	X	X	-	-	-	(X)
Generate events	X	-	X	X	X	-	X	X	X	X
Modify model state	-	-	X	X	X	n/a	X	-	X	(X)
Automated Control of Simulation										
(User-defined) Scripts	(X)	-	-	-	-	-	-	X	-	X
Replay of prev. runs	(X)	-	-	X	-	n/a	-	-	X	X
Animation										
Graphical notation(s)	X	X	X	X	X	-	X	X	X	X
Customisable GUI	-	-	-	-	(X)	-	-	-	-	X
External GUI	X	-	X	X	X	-	-	X	X	X
Anim. real-time	-	n/a			-	-	-	X	-	X
Anim. of code fragments	n/a	n/a	-	n/a	-	n/a	n/a	-	n/a	n/a
Recording simulations										
Traces/Logs	(X)	(X)	X	X	-	-	(X)	X	X	(X)
Statistics	-	-	-	-	X	-	-	-	-	X
Time lines	-	X	-	-	-	-	X	-	-	-
Validation										
Model coverage views (per single sim. run)	-	(X)	-	X	-	-	-	-	X	X
Validation management	-	-	-	-	-	-	-	-	-	-

Table 3: Simulation/animation capabilities (1)

	CIP Tool 3.0	COREsim 2.1	ObjecTime Developer 5.0	ObjectGEODE 4	PACE 3.1	QuickCRC 1.2	RDD-100 / DVF 4.1	Rhapsody for C++ 2.1	SDT / ORCA 3.3	Statemate MAGNUM 1.3
Verification										
Syntax	X	X	X	X	X	n/a	X	X	X	X
Statically checked properties	X	X	X	X	X	n/a	X	X	X	X
… programmable	-	-	-	-	-	-	-	-	-	-
Dynamically checked properties	-	-	-	X	-	n/a	X	-	X	X
… programmable	-	-	-	X	-	-	-	-	-	-
Invariants	-	-	-	X	-	-	-	-	-	-
Compare “MSC” against model	-	-	X	X	-	-	-	(X)	X	-
Random exploration	-	-	-	X	-	-	-	-	X	-
Limited depth exploration	-	-	-	X	-	-	-	-	X	-

Table 4: Simulation/animation capabilities (2)

5. The tools not surveyed

This chapter lists all the tools that we initially had taken into account, but that did not met the criteria that define the scope of the survey (as defined in section 3.1), or that were not further evaluated for some other reason (e.g. limited resources). The tools not surveyed are grouped into the following categories, according to the reason of exclusion:

- Tools without interactive simulation and animation
- Tools without RE/SE modelling language support
- Lack of information about tools
- Lack of time for further investigation of tools
- Discontinued tools

5.1 Tools without interactive simulation/animation capabilities

Table 5 – Table 7 list the tools that were not equipped with a "RE-suitable" and interactive simulator/animator, as defined in section 3.1, up to the point of time, when we stopped further elicitation of information.

Category: Requirements Acquisition, -Analysis, -Management / CAE

Tool name	Date of decision / Last update
Cradle SEE	03.06.1997
DOORS	25.03.1997
PC Pack	04.11.1997
ProductTrack	27.08.1997
RA	20.04.1998
RATS tool	23.03.1998
RDT	03.06.1997
Requisite Baseline RequisitePro	03.06.1997
RTM	03.06.1997
SLATE Architect	03.06.1997
TMA toolset	04.11.1997
Tracer	04.11.1997
Vital Link	25.03.1997
XTie-RT	03.06.1997

Table 5: Tools without Interactive Simulation/Animation Capabilities (1)

Category: Workflow / Business Process (Re-)Engineering

Tool name	Date of decision / Last update
Arena Standard Edition etc.	25.03.1997
IvyFrame	19.03.1999

Table 6: Tools without Interactive Simulation/Animation Capabilities (2)

Category: CASE

Tool name	Date of decision / Last update
BetterState Lite	07.09.1998
Innovator	27.08.1997
LOCANA	27.08.1997
METIS 97	04.11.1997
Objects9000	24.03.1998
ObjectTeam	18.12.1998
Melba 96 / X Melba	03.06.1997
ROSE	03.06.1997
StP / UML	18.06.1999
TeamWork	04.11.1997
ticketOstar	26.05.1998
Win A&D, Mac A&D	06.03.1998

Table 7: Tools without Interactive Simulation/Animation Capabilities (3)

Comments:

- *BetterState Lite* provides merely back animation, but no simulator according to section 3.1. *BetterState Lite* is based on a Statechart-like language for modelling software systems. It generates code (C++, VHDL, Perl, …) from the models, which can be compiled and executed. The code is instrumented in order to record state transitions in a file or database. This recording can be used later to back-animate the model, after running the generated code. But there is no interactive simulator.
- Some tools of Table 7 are nevertheless suited for rapid prototyping.

5.2 Tools without RE/SE modelling language support

The tools listed in Table 8 are mainly general purpose simulation tools that support other modelling languages (e.g. equation systems) than typically used in RE/SE. In the context of RE, these modelling languages are hardly an appropriate substitute for executable specifications or prototyping. That's why the tools of this category were excluded from the survey.

Tool name	Date of decision / Last update
ISLE	03.06.1997
MicroSaint	25.03.1997
MODSIM III	03.06.1997
SimulationExpert	27.08.1997
SIMPRO	27.08.1997

Table 8: Tools without RE/SE Modelling Language Support

Comments:

- More tools of this kind can be found at "http://ws3.atv.tuwien.ac.at/comparisions/".

5.3 Lack of information about tools

Concerning the tools listed in Table 9, it was extremely difficult to obtain detailed information within a reasonable time frame. Most of them stemmed from research projects that already had been discontinued when we started our survey. We decided to exclude these tools, since a (more or less) fair comparison was not possible.

Tool name	Date of decision / Last update
CAT	19.10.1999
EGS 2	19.03.1999
MacroTec	07.09.1998
REVS	07.09.1998
RSML Simulator	06.03.1998
SITE	19.03.1999

Table 9: Tools we had to exclude, because we could not obtain enough information.

Comments:

- Surprisingly the *RSML Simulator* had to be excluded from this survey. *RSML Simulator*, as its name indicates, actually provides a simulator / animator. Since RSML is a Statechart-based modelling language, animation of a model comes quite naturally. Nevertheless, our requests for detailed information or a demo version of *RSML Simulator* were not answered, so we decided to exclude this tool.
- Furthermore, *RSML Simulator* was supposed to be discontinued and replaced by *SpecTRM-RC*.

5.4 Lack of time for further investigation of tools

Due to the large number of candidate tools but limited resources, we decided to exclude tools where information gathering took too much effort (see Table 10).

Tool name	Date of decision / Last update
ASADAL / SIM	19.03.1999
AutoFocus	19.03.1999
Design / CPN	19.03.1999
Possum	09.07.1999
VeriSpec	09.07.1999

Table 10: Tools we had to exclude for lack of time to investigate on them in more detail.

6. Conclusions

We have presented a survey of ten requirements engineering tools with capabilities for simulation and animation of models. Each tool has been evaluated with respect to seven aspects: execution, interactive and automated control of simulation, animation, recording, validation, and verification. The main results have been presented in Table 3 and Table 4 on pages 26 and 27.

In general, we observed that the simulation capabilities are very similar among the evaluated tools. The very details of the simulation/animation capabilities of any tool strongly depend on the supported modelling languages. We observed the most striking differences in the area of support of validation and verification.

Below, we present some findings based on the survey results and the experience we made during the survey.

General observations/experiences - Since we started with a broad search of simulation and/or RE tools, we initially considered many tools that were not directly suited for REthat were not model-based or that did not feature simulation/animation capabilities. The remaining candidates were either CASE or system engineering tools, mostly stemming from embedded systems' development.

It was partially cumbersome and tedious to obtain information (brochures, white papers, manuals or demo versions) about the tools. We experienced a high diversity of customer-friendliness and responsiveness when contacting the corresponding tool vendors or distributors.

In general, the surveyed tools (demo versions, to be precise) were found to be quite intuitive to use.

Execution capabilities - Code generation usually implies that various platforms are supported, due to the high availability of corresponding cross-compilers. On the other hand, interpretation is limited to the platformsthat are supported directly by the simulation tool. PACE is the only interpreter-based tool of the survey that offers a virtual machine in order to execute Petri Nets on a remote platform.

ObjecTime Developer, ObjectGEODE and Statemate MAGNUM (3 of 10 tools) provide direct interpretation as well as code generation.

Interactive/automated control capabilities - Forward stepping (of various kinds) is supported by all of the 10 surveyed tools. CIP Tool and QuickCRC provide just one sort of (single) step, while the other tools provide various kinds of forward stepping. In contrast, backward stepping is not so widespread (4 of 10 tools). Statemate MAGNUM allows to go back one step.

Most tools support the injection of events/messages. The only kind of events supported by QuickCRC is triggering scenarios. COREsim runs the simulations automatically, guided by computation of resource allocation (not by user-controlled events).

Automated control of simulation runs is generally not supported. Just CIP Tool and Statemate MAGNUM are able to (re-) run both user-defined and recorded scripts.

Animation capabilities - All tools, except QuickCRC, support modelling languages with graphical notations that are animated.

Besides animating the graphical notations supported, PACE and Statemate MAGNUM (2 of 10 tools) additionally support customisable GUIs. In contrast to Statemate MAGNUM, PACE offers only few GUI elements that allow to directly modify the state of the system model. All the toolsthat are based on code generation (6 of 10 tools) allow to link separately programmed GUI code to the generated code.

Animation in real-time is only supported by 2 of 10 tools: Rhapsody and Statemate MAGNUM.

None of the three tools that require to formulate execution details in a different (standard) programming language is able to animate such code fragments.

Recording capabilities - Simulation traces/logs are supported by 8 of 10 tools. Four of them record simulation runs in a format (e.g. MSC) that a tool user easily can read or even modify.

Only PACE and Statemate MAGNUM support computation and representation of statistics about the system model.

Time lines are supported by two of 10 tools: COREsim and RDD-100 /DVF, the two system engineering tools in the survey.

Validation capabilities - ObjectGEODE, SDT/ORCA and Statemate MAGNUM support model coverage per simulation run. But no tool offers any kind of validation management/support that reaches beyond this.

Verification - All tools, except QuickCRC (due to the supported CRC modelling language), do syntactic and semantic checks prior to simulation runs. None of the surveyed tools provides user-definable (i.e. programmable) semantical checks.

Checking of dynamic properties (i.e. semantic checks at simulation time) are supported by ObjectGEODE, SDT/ORCA and Statemate MAGNUM. ObjectGEODE even supports programmable checking of dynamic properties.

Just one of 10 tools (ObjectGEODE) supports invariants.

ObjecTime Developer, ObjectGEODE and SDT/ORCA allow to compare an MSC against a system model. Rhapsody allows to compare two MSCs.

Random and limited-depth state space exploration are supported by ObjectGEODE and SDT/ORCA, the two SDL-based tool of the survey.

In general, ObjectGEODE and SDT/ORCA encompass the broadest range of verification functions among the surveyed tools.

In general, we observed that the simulation capabilities are very similar among the evaluated tools. The very details of the simulation/animation capabilities of any tool strongly depend on the supported modelling languages. We observed the most striking differences in the area of support of validation and verification.

7. References

7.1 Literature

[Berner et al. 98]
Berner, Stefan; Joos, Stefan; Glinz, Martin. “A Visualization Concept for Hierarchical Object Models”. Proceedings of the 13th IEEE International Automated Software Engineering Conference (ASE 1998), Honolulu, Hawaii. Washington, etc.: IEEE Computer Society, Oct. 1998, pp. 225-228.

[Bloss et al. 92]
Bloss, Felix; Willutzki, Paul; Adiprasito, Bartano; Paulini, Michael. “Automatische Modellgenerierung zur Simulation der Kommunikationsumgebung im Kraftfahrzeug”. it + ti – Informationstechnik und Technische Informatik, Oldenburg Verlag, Vol. 41, No. 3, 1999, pp. 29-35.

[Boehm 81]
Boehm, Barry W. “Software Engineering Economics”. Prentice-Hall, 1981, ISBN 0-13-822122-7.

[Budde et al. 92]
Budde, R.; Kautz, K.; Kuhlenkamp, K.; Züllighoven, H. “Prototyping: An Appraoch to Evolutionary System Development”. Springer Verlag, 1992, ISBN 3-540-54352-X.

[Hazel et al. 97]
Hazel, Daniel; Strooper, Paul; Traynor, Owen. “Possum: An Animator for the SUM Specification Language”. Proceedings Asia-Pacific Software Engineering Conference and International Computer Science Conference, IEEE Computer Society, Dec. 1997, pp. 42-51.

[IEEE 90]

The Institute of Electronical and Electronics Engineers. “Standard Glossary of Software Engineering Terminology”. IEEE Std 60.12-1990, IEEE Computer Society Press, 1990.

[Leveson et al. 96]
Leveson, Nancy G.; Heimdahl, Mats Per Erik; Hildreth, Holly; Reese, Jon Damon. “Requirements Specification for Process-Control Systems”. IEEE Transactions on Software Engineering, Vol. 20, No. 9, Sep. 1994, pp. 684-707.

[McDermid 94]
McDermid, John A. “Software Engineer’s Reference Book”. Butterworth-Heinemann Ltd., 1994, ISBN 0-7506-0813-7

[Kang et al. 98]
Kang, Kyo C.; Lee, Kwan W.; Lee, Ji Y., Kim Gerald J. “ASADAL/SIM: An Incremental Multi-level Simulation and Analysis Tool for Real-time Software Specifications”. Software – Practice and Experience, Springer-Verlag, Vol. 28, No. 4, Apr. 1998, pp. 445-462.

[Gaskell et al. 94]
Gaskell, Craig; Phillips, Roger. “Executable Specifications and CASE”. Software Engineering Journal, Jul. 1994, pp. 174-182.

[Reubenstein et al. 91]
Reubenstein, Howard B.; Waters, Richard C. “The Requirements Apprentice: Automated Assistance for Requirements Acquisition”. IEEE Transactions on Software Engineering, Vol. 17, No. 3, Mar. 1991.

[Selic et al. 92]
Selic, Bran; Gullekson, Garth; Ward, Paul T. “Real-Time Object-Oriented Modelling”. John Wiley & Sons, Inc., 1994, ISBN 0-471-59917-4.

[Wieringa 98]
Wieringa, Roel. “A Survey of Structured and Object-Oriented Software Specification Methods and Techniques”. ACM Computing Surveys, Vol. 30, No. 4, Dec. 1998, pp. 459-527.

[Wirfs-Brock et al. 90]
Wirfs-Brock, Rebecca; Wilkerson, Brian; Wiener, Lauren. “Designing Object-Oriented Software”. Prentice Hall International Inc., 1990, ISBN 0-13-202664-3.

7.2 Tools

This paragraph lists references for all the tools mentioned in this document, in alphabetical order.

Arena® Standard Edition, etc.
Systems Modeling Corporation
Sewickley, Pennsylvania, USA
http://www.sm.com/

ASADAL / PROTO, ASADAL / SIM
Pohang University of Science and Technology
Korea
[Kang et al. 98]
ASADAL = “A System Analysis and Design Aid tooL”

AutoFocus
Chair of Prof. Manfred Broy
Department of Computer Science
Technical University of Munich (TUM)
München, Germany
http://autofocus.informatik.tu-muenchen.de/

BetterState Lite
Integrated Systems, Inc. (ISI)
Sunnyvale, California, USA
http://www.isi.com/

CAT
CAT is part of the CoRE tool set
British Aerospace
GB
CAT = “CoRE Animation Tool”
CoRE = “Controlled Requirements Expression”

CIP Tool®
CIP SYSTEM AG
Zuerich, Switzerland
http://www.ciptool.ch/
CIP = “Communicating Interacting Processes”

COREsim
COREsim is part of the CORE tool set
Vitech Corporation
Vienna, Virginia, USA
http://www.vtcorp.com/

Cradle® SEE
Structured Software Systems Limited (3SL)
Olney, Maryland, USA
http://www.threesl.com/
SEE = “System Engineering Environment”

Design/CPN
University of Aarhus
Aarhus, Denmark
http://www.daimi.au.dk/designCPN/
CPN = “Coloured Petri Net”

DOORS®
Quality Systems & Software Limited
Mt. Arlington, New Jersey, USA
http://www.qssinc.com/
DOORS = “Dynamic Object-Oriented Requirement Specification”

EGS 2
University of Hull
UK
[Gaskell et al. 94]
EGS = “Executable Graphical Specification”

ISLE
Research in Artificial Intelligence and Software Engineering (RAISE Lab)
Brigham Young University
Utah, USA
http://slice.nosc.mil/coaster/isle/doc/ISLE94/ISLE/ISLE.html
ISLE = “Integrated Simulation Language Environment”

Innovator
MID GmbH
Hamburg, Germany
http://www.mid.de/

IvyFrame
IvyTeam
Zug, Switzerland
http://www.ivyteam.com/

LOCANA
Rapid Prototyping Laboratory (RPL)
Software Engineering Group
University of Sunderland
Sunderland, UK
http://osiris.sunderland.ac.uk/rpl/
http://osiris.sund.ac.uk/
LOCANA = “Loosely Orthogonal Class-Activity Notation for Analysis”

MacroTec
Software Engineering Group (GELO)
University of Montreal
Montreal, Canada
http://www.iro.umontreal.ca/labs/gelo/

Melba 96 / X Melba
Center for Advanced Technology in Telecommunications (CATT)
Royal Melbourne Institute of Technology (RMIT)
Melbourne (Victoria), Australia
http://www.catt.rmit.edu.au/catt/

METIS 97
METIS Solutions / NCR Norway AS
Horten, Norway
http://www.metis.no/

MicroSaint
MICRO ANALYSIS & DESIGN, Inc. (MA&D)
Boulder, Colorado, USA
http://www.maad.com/

MODSIM III®
CACI International, Inc.
Arlington, Virginia, USA
http://www.caciasl.com/modsim.html/
MODSIM = “Modular Simulation language”

ObjectGEODE™
VERILOG S.A.
Toulouse, France
http://www.csverilog.com/

ObjecTime® Developer
ObjecTime Limited
Kanata, Canada
http://www.objectime.com/
[Selic et al. 92]

Objects9000
Roesch Consulting GmbH
Kaarst, Germany
http://www.roesch.com/

ObjectTeam
(meanwhile called ”COOL: Jex”)
Sterling Software, Inc. (formerly Cayenne Software, Inc.)
Plano, Texas, USA
http://www.sterling.com/

PACE
IBE Software und Simulation Engineering
Glonn, Germany
http://ourworld.compuserve.com/homepages/ibepace/

PC Pack
Integral Solutions Limited (ISL)
USA
http://www.isl.co.uk/pc_pack.html

Possum
Software Verification Centre
Department of Computer Science and Electrical Engineering
University of Brisbane
Brisbane, Australia
http://www.svrc.it.uq.edu.au/Pages/Animation.html
[Hazel et al. 97]

ProductTrack
Teknowledge Corp.
Palo Alto, California, USA
http://www.teknowledge.com/

QuickCRC™
Excel Software
Marshalltown, Iowa, USA
http://www.excelsoftware.com/
CRC = "Class Responsibility Card"

RA
[Reubenstein et al. 91]
RA = "Requirements Apprentice"

RATS tool
A.Eberlein, F. Halsall
Department of Electrical & Electronic Engineering
University of Wales
Swansea, UK
http://kona.swan.ac.uk/~eeeberle/
RATS = "Requirements Assistant for Telecommunication Services"

RDD-100® / DVF™
Ascent Logic Corporation
San Jose, California, USA
http://www.alc.com/
RDD = "Requirements Driven Development"
DVF = "Dynamic Verification Facility"

RDT®
GEC Marconi Systems Pty (Australia)
…
http://www.gec-marconi.com/
RDT = "Requirements Database Tool"

RequisitePro®
Rational Software Corp.
Cupertino, California, USA
http://www.rational.com/

REVS
McAllford
REVS = "Requirements Engineering Validation System"

Rhapsody™
I-Logix, Inc.
Andover, Massachusetts, USA
http://www.ilogix.com/

ROSE®
Rational Software Corp.
Cupertino, California, USA
http://www.rational.com/

RSML Simulator
University of Washington
USA
http://www.cs.washington.edu/research/projects/safety/www/papers/kurtquals/node1.html
[Leveson et al. 94]
RSML = “Requirements State Machine Language”

RTM®
Integrated Chipware, Inc. (formerly Marconi Systems Technology, Inc.)
Reston, Virginia, USA
http://www.chipware.com/
RTM=“Requirements & Traceability Management”

SDT / ORCA
SDT and ORCA are part of Telelogic Tau™
Telelogic SA
Sweden
http://www.telelogic.se/
SDT = “SDL Design Tool”
ORCA = “Object oriented Requirements Capture and Analysis”

SIMPRO
Department of Physics and Astronomy
University of Hawaii
Honolulu, Hawaii, USA
http://www.phys.hawaii.edu/

SimulationExpert®
Visual Logic
USA
http://www.visual-logic.com/

SITE
Systems Analysis
Humboldt University of Berlin
Berlin, Germany
http://www.informatik.hu-berlin.de/Institut/struktur/systemanalyse/SITE/SDL-tools.html
SITE = “SDL Integrated Tool Environment”

SLATE™ Architect
TD Technologies Inc.
USA
http://www.tdtech.com/
SLATE = “System Level Automation Tool for Enterprises”

Statemate® MAGNUM™
I-Logix, Inc.
Andover, Massachusetts, USA
http://www.ilogix.com/

StP® / UML
AONIX
USA
http://www.aonix.com/
StP = “Software through Pictures”
UML = “Unified Modelling Language”

TeamWork
(meanwhile called “COOL: Teamwork”)
Sterling Software, Inc. (formerly Cayenne Software, Inc.)
Plano, Texas, USA
http://www.sterling.com/

ticketOstar
Christian Clercin
Groupe ARIAL
Ecole Nationale d'Informatique
Universite de Fianarantsoa
Madagaskar
http://www.refer.mg/madag_ct/edu/fianar/eni/ticketos.htm

TMA toolset
System Engineering & Assessments SEA Ltd.
USA
http://www.naval-technology.com/contractors/project/sea/index.html

Tracer
ISDE Limted
USA
http://www.isde.com/tracer/index.html

VeriSpec
CACI Advanced Simulation Lab
CACI Products Company
Arlington, Virginia, USA
http://www.caciasil.com/

Vital Link
Compliance Automation, Inc.
USA
http://tlmworks.com/cai

Win A&D, Mac A&D
Excel Software
Marshalltown, Iowa, USA
http://www.excelsoftware.com/

XTie-RT®
Teledyne Brown Engineering
Huntsville, Alabama, USA
http://www.tbe.com/

8. Glossary

2D	2-dimensional
4GL	4th Generation Language
ASN.1	Abstract Syntax Notation 1
BPR	Business Process Reengineering
CAE	Computer-Aided Engineering
CASE	Computer-Aided Software Engineering
CPN	Coloured Petri Nets
CRC	Class Responsibility Collaboration
DFD	Data Flow Diagram
ERM	Entity Relationship Model
FSM	Finite State Machine
GI	German Information society
GUI	Graphical User Interface
IEEE	Institute of Electronical and Electronics Engineers
INCOSE	International Council on Software Engineering
ITU	International Telecommunication Union
MSC	Message Sequence Charts
OML	Object Modeling Language
OOA	Object-Oriented Analysis
OOD	Object-Oriented Design
Pr/T Nets	Predicate/Transition Nets
RE	Requirements Engineering
SA / RT	Structured Analysis / Real-Time
SA / SD	Structured Analysis / Structured Design
SDL	Specification and Design Language
SE	System Engineering/Software Engineering
TTCN	Tree and Tabular Combined Notation
UML	Unified Modeling Language
VCR	Video cassette recorder
VHDL	Verilog Hardware Description Language

Appendix A: Preliminary questionnaire

This appendix presents the preliminary questionnaire that we used for structuring the information about the tools we found during the broad search. This questionnaire served as a basis for deciding whether to include a tool into the survey or not.

Author	- Name of the author, evaluating the tool
"Company" Name	- ...
"Company" Location	- ...
Product Name	- ...
Product Release	- ...
Product Status	- research prototype - commercial, new product - commercial, established single product - commercial, established product family
Product Prices	- …
Supported Platforms	- Unix: ... - Windows 95 - Windows NT - MacIntosh - OS/2
Tool class	- CASE - Discrete Event Simulation - Requirements Management - RE Tool
Supported Methods	- Booch (OOD) - Rumbaugh (OMT) - Jacobson (OOSE) - Coad / Yourdon (OOA) - Jackson (JSD) - others: ... - proprietary: ...
Supported Languages	- SDL (Specification and Design Language, ITU Z.100) - MSC (Message Sequence Charts, ITU Z.120) - ASN.1 (Abstract Syntax Notation 1) - UML (Unified Modelling Language) - ERM (Entity Relationship Model) - DFD (Data Flow Diagram) - others: ... - proprietary: ...
Basic Paradigms	- Petri Nets - Statecharts - OO-Model - SA / SD - others: ... - proprietary: ...
Structure Model	- Statecharts - OO-Model - SA / SD - others: ... - proprietary: ...
Behaviour Model	- State Charts - (extended) FSM - Petri Nets - MiniSpecs - others: ... - proprietary: ...

Data Model	- ERM - OO-Model - Data Dictionary - ASN.1 - others: ... - proprietary: ...
Timing Model	- synchronous (broadcast-message) - asynchronous - mixed
Simulator / Animator	- yes, integrated simulator / animator - yes, separate simulator / animator - code generator + specialised debugger - code generator - no
Typical Usage	- Telecommunication - Automation - Financial software - others: ...
Comment	- any comment
Last Update	- any date

Appendix B: Detailed questionnaire

This appendix contains the detailed questionnaire that we used for structuring the information we gathered about the tools that matched the project's scope. This questionnaire served as a basis for the evaluation and comparison of the selected tools.

The questionnaire is divided into several sections. Each section has a title and a table, containing the information about the tool. Each table has three columns. The first column "Attribute" defines the kind of information. The second column "Info" contains a short code (see legend at the end of the questionnaire) denoting the source(s) of the information, plus a reference number. The third column "Domain" contains the actual information. Each field of this column contains a marker, denoting if the information is an enumeration ("[enumeration type]", i.e. only one of the predefined answers must be selected) or a set type ("[set type]", i.e. any combination of the predefined answers is allowed), and a predefined list of expected answers. Some of the questionnaire attributes are marked to require a detailed description, not just keywords.

All the used referential materials like web pages, brochures, manuals, contact persons, demonstrations and so on, had to be numbered, had to be filled into the column "Info" and had to be listed at the end of the questionnaire, so the traceability of the information to its sources could be guaranteed.

Tool identification

Attribute	Info	Domain
Author	...	- name of the author, who collected the tool information
Vendor (name)	...	- …
Vendor (location)	...	- …
Product name / release	...	- …
Last update	...	- … last update of the questionnaire

Modelling Basics

Attribute	Info	Domain
Supported languages / notations	...	[set type] [… add detailed description !] - OML (Object Modelling Language) - SA (Structured Analysis) - SA / RT (Structured Analysis / Real Time) - SDL (Specification and Design Language) - Petri Nets - UML (Unified Modelling Language) - Z - other: ... - proprietary: ...
Basic paradigms of simulation (execution)	...	[set type] [… add detailed description !] - automaton (FSM, Statecharts) - DFD - MSC, Object Message Diagram - Petri Nets - other: ...
Formality degree of the models	...	[enumeration type] [… add detailed description !] - informal, textual - semiformal - strictly formal

Specification / definition of the execution semantics (including detail semantics)	...	[enumeration type] - implicitly defined by the simulator - explicitly defined, with direct references to corresponding literature - explicitly defined in the manuals
Access to meta model	...	[enumeration type] - yes, ... - no
Supported abstraction mechanisms	...	[set type] [… add detailed description !] - aggregation - delegation - inheritance - other: ...
structure / architecture model	...	[enumeration type] [… add detailed description !] - aggregation / hierarchy ...
timing model of messages	...	[set type] [… add detailed description !] - synchronous (broadcast messages) - asynchronous - both
Timing model of / access to data blocks	...	[enumeration type] [… add detailed description !] - with locking mechanisms ("atomic" access) - without locking mechanisms

Simulator- / animator features

Attribute	**Info**	**Domain**
Code generation - 1.	...	[enumeration type] - yes, optional - yes, needed for simulation - no
Code generation - 2.	...	[set type] - generates complete code ; and that for languages ... - generates code frame ; and that for languages ...
Recording / tracing and playback of simulation runs	...	[enumeration type] - yes - no
Simulation of the system environment / actors	...	[enumeration type] - none - same possibilities as for the model - stochastic modelling (casualty, unpredictability) - other: ...
Supported analyses	...	[set type] - traces / logs - coverage - other: ...
Verifications	...	[enumeration type] - yes, ... - no
Test case derivation (for system testing)	...	[enumeration type] - yes, ... - no
Execution of incomplete models	...	[enumeration type] - yes, an that ... - no

Control of the simulation runs	...	[set type] - all paths - single path - interactive - to the first unexecutable statement - controlled by test suite - other: ...
Simulated time	...	[enumeration type] - continuous simulated "real time" (acceleration / deceleration ??); i.e. for execution and animation - simulated "real time" for execution, but not for animation - global simulation time no supported
Debugging features	...	[set type] - breakpoints - watch points / inspectors - others:...

Applications

Attribute	Info	Domain
Simulation purpose	...	[set type] - feasibility study - communication means (RE-Engineer <--> RE-Engineer) - communication means (RE-Engineer <---> customer / user) - validation of functionality (correctness, completeness, ...) - performance analysis: general timing behaviour - performance analysis: throughput - performance analysis: usage
Typical application area	...	[set type] - IS applications - real time systems / embedded systems - processes (e.g. Workflow Systems) - system internal services / middleware

Marketing / Distribution

Attribute	Info	Domain
Status - 1.	...	[enumeration type] - commercial - research - proprietary (internally developed and used)
Status - 2.	...	[enumeration type] - prototype - announced for ... - newly available - established (available)
Status - 3.	...	[enumeration type] - single product (software) - family of fitting products
Supported platforms	...	[set type] - Unix: Solaris, HP-UX, AIX, ULTRIX, Linux ... - Windows 95 - Windows NT - MacOS - OS/2
Pricing (list prices)	...	[set type] - Demo version: ... - Educational version: ... - Commercial version: ...

Official product strategy	...	[enumeration type] - discontinue - keep it as it is - extend - merge with product ...

Add-On's

Attribute	Info	Domain
Exports of models	...	[enumeration type] - yes, and that... - no
Imports of models	...	[enumeration type] - yes, and that... - no
Interoperability	...	[enumeration type] - integrated with products ... (i.e. based on a common repository / DB) - not integrated
Version control of models or model components	...	[enumeration type] - integrated, bundled - additional product of same supplier - additional product of other supplier - not supported
Multi-user access, access management	...	[enumeration type] - yes, configurable ... - no
Available libraries, components, frameworks, ...	...	[enumeration type] - yes, and that ... - no

Experiences with the tool (demo version)

Attribute	Info	Domain
Comprehensibility of model representation	...	[enumeration type] - too much per view, and that ... - satisfactory - too less per view, and that ...
Userfriendlyness: ease of learning	...	[enumeration type] - simple, intuitive (manuals are rarely needed) - medium (can be mastered with manuals), and that ... - difficult (training and support needed), and that ...
Userfriendlyness: ease of use	...	[enumeration type] - simple, intuitive (manuals are rarely needed) - medium (can be mastered with manuals), and that ... - difficult (training and online support needed), and that ...
Stability	...	[enumeration type] - no problems - casual crashes / bugs, and that ... - several crashes / bugs, and that ...
Performance	...	[enumeration type] - allows to work quickly - repeatedly waiting periods up to ...
Documentation	...	[set type] - satisfactory, because ... - not enough information, and that... - information was difficult to find, and that ... - ...

Support: technical questions	...	[enumeration type] - satisfactory, because ... - unsatisfactory, because ...
Support: training	...	[enumeration type] - satisfactory, because ... (competence, course schedules, course materials, ...) - insufficient, because ...

Legend of information sources:

(used in the columns "Info")

web = web pages
mkt = marketing materials (brochures)
man = manuals
con = contact to other users of corresponding tool
rep = experience report provided by other users of corresponding tool
dem = live demonstration by vendor / distributor
ref = tested with own reference model (ticketing system)
ral = tested with alternative reference model (...)
dir = directly experienced with the tool (not needing some reference model at all)
lit = literature (papers, books, …)