An Evaluation of High-End Tools for Petri-Nets

Harald Störrle

Ludwig-Maximilians-Universität München
Institut für Informatik
Bericht 9802
June 1998

Contents

1 Preliminaries **3**
1.1 Motivation . . . 3
1.2 Why Petri-Nets? . . . 3
1.3 How to get a tool . . . 4

2 The selection process **6**
2.1 The first round: eliminating unfit tools . . . 6
2.2 The second round: testing tools . . . 7
2.3 The third round: classifying and comparing tools . . . 8

3 Round 1: eliminating unfit tools **8**

4 Round 2: testing tools **9**
4.1 General information . . . 11
4.1.1 Providers, Packages, Prices . . . 11
4.2 Class of Petri-nets . . . 11
4.3 Functionality . . . 16
4.3.1 Integration and convenience . . . 16
4.3.2 Graphical Editor . . . 21
4.3.3 Simulator . . . 23
4.3.4 Analysis tools . . . 26
4.4 Maintenance, Documentation, and future plans . . . 28
4.5 Openness and interfaces . . . 30

5 Round 3: comparing and classifying tools **32**
5.1 Usage profile for relative rating . . . 33
5.2 Classification of tools . . . 33
5.3 Further information . . . 37
5.4 Recommendations . . . 37

6 Summary and outlook **37**

Version: 1.2 of August 3, 1998

Abstract

Petri-nets are a well-known formalism to model dynamic aspects of systems. They have found many applications over the last decades and are increasingly popular for greatly varying purposes. Paramount for any real-life application (and even for realistic case-studies) is a computer based tool to support the use of Petri-nets. There is a large number of tools already available, but no detailed survey comparing them. The purpose of the study described in this paper was to evaluate them, and thus make possible an informed choice of tools.

After some preliminary remarks on my motivation, the requirements and the selection process, I present a detailed analysis of the relevant features of the tools. These results are compared and recommendations derived from them.

1 Preliminaries

In this section I first describe the main motivation that led me to conduct this study. From it, a number of requirements for a tool arise, that determine the selection process.

1.1 Motivation

Many Software-Engineering methodologies used in industry lack a sound formal basis. This applies, for instance, to the Unified Modeling Language (UML, [3]), and to Object-Oriented Software Engineering after Jacobson (OOSE, [12]). Recent work (see [20, 17, 23] for foundational works, and [18] for a case study) has shown, that hierarchical high-level Petri-nets are quite suitable as such a basis.

Obviously, developing a practical methodology needs to be guided by practical experience if it is to be of any relevance. To conduct case-studies and projects of realistic size, however, tool-support is indispensable. So, this evaluation was conducted to select a tool for hierarchical high-level Petri-nets meeting our requirements.

Rather than using Petri-nets as a modelling languages themselves (see [22]), with a view to replacing existing notations such as UML, it is more realistic to provide these notations with a formal semantics expressed in Petri-nets. This approach allows the adoption and integration of basically any notation. This is particularly important if one is concerned with the maintenance of software systems: Any obscure ad-hoc notations that might have been used in the documentation of the system can, in principle, be integrated, as long as it is structured and more or less consistent. Note, that this way, the system knowledge of the people involved with the system's maintenance is much more easily accessible than by trying to replace whatever formalism is used by some notation new to these people. This is strictly more important than being able to exploit some minor advantage that one formalism might have over another. Note also, that the approach is strictly neutral to the methodology or notation a system has been designed or implemented in. This project is described in greater detail in the forthcoming technical report [23].

Apart from this, a general evaluation and comparison of Petri-net based tools is of considerable interest to the Petri-net community, as the amount of replies to a recent posting in the Petri-Net-Newsletter shows. In spite of this interest, no such general evaluation seems to exists. Given the number of tools available, and the functionality they offer, it is quite clear that such a general study is beyond the scope of this survey, simply for reasons of economy.

1.2 Why Petri-Nets?

The ultimate goal of the approach described above is to provide a formal basis for software-engineering for distributed systems with a view to support software maintenance in particular by translating the various models (or 'views') of a system as given for instance by UML diagrams into a common formalism. Thus, a formal analysis and formally based checks of consistency of a compound model become possible. There are a number of conflicting requirements that arise for this common formalism:

1. it must capture both static and dynamic aspects of a system;
2. it must accommodate for state- as well as for data- and event-based modelling styles in a natural way;
3. it must allow for the integration of many (all) modelling languages, most of which are not known at the point of selecting the formalism[1];
4. it must be capable of representing systems on multiple levels of abstraction.

Point number 4 is very important: in practical applications, one is often confronted with some ad-hoc or application-specific notation. It is starkly unrealistic to adhere to any set of modelling languages for all systems and all purposes, and all types of people involved. Also, for almost all systems it is simply too expensive to transform existing documentation from some notation, however arcane it might be, into some other formalism – the language used with that system is the language that needs to be used with any tools to support maintenance of the system.

Apart from checking consistency among models, debugging and validating single models is also a necessity. Thus, the following is also required from the common formalism:

5. it must lend itself to simulation and animation;
6. there need to be efficient analysis techniques available to check properties of models;
7. there should be tools available to deal with it.

In another paper (see [23]), I propose a new form of high-level Petri-nets called Procedural Petri-nets (PPNs) as the common formalism (other approaches using stream-processing functions or various kinds of finite-state automata also exist). Obviously, Petri-nets fulfil the requirements 1, 2, and 5. In separate papers, I show that PPNs are indeed suitable to model hierarchically structured systems (req. 4), that they are efficiently analysable (req. 6), and that there is a wealth of highly developed Petri-net tools available (req. 7). As for the third requirement, there are several works on how common software-engineering notations may be translated into (various forms of) Petri-nets and how the advantages of Petri-nets may be exploited (see for instance [20, 2, 15, 19, 23]).

1.3 How to get a tool

Basically, there are three ways of obtaining an integrated tool featuring a graphical editor, a simulator, and analysis tools: either one can program it oneself, or one can add and integrate a set of specialised tools, or one can adapt and enhance an already existing integrated tool.

1. *program oneself*
   While this approach guarantees that all requirements will be met, it incurs

[1] This point is essential in re-engineering applications, where all sorts of documents need to be integrated, including documents in proprietary notations of the customer.

enormous effort in terms of programming. The burden might be alleviated by using a suitable framework like the Petri-Net Kernel ([14, 26]) or GraphFrame ([21]).

Both of these, however[2], do not provide enough support to render this approach a viable one:

(a) GraphFrame is focused on general graphs. While supporting rich editing facilities, no provision is made for analysis.

(b) The PNK is yet in a very early stage of its development (the currently available version is 0.8), and the abstractions it provides are not yet sufficient to build a large integrated tool on top of it, with the functionality required.

So, while the use of a framework might become an interesting option in the future, there are currently no sufficiently well-developed approaches.

2. *integrate specialised tools*
   In the scientific community, there is a number of well-known specialised analysis like INA, and PROD. They could be integrated with a graphical editor to form a workbench. Suitable front-ends might be the Petri-net Editor PED[3], the generic graphical editor that is to be produced as a result of the Diploma-thesis by Z.Gassmann, or a specifically enhanced version of `xfig` such as that of B.Meyer.

   In this approach, some programming effort will have to go into the integration of the tools. Typically, all tools use individual formats, most of the time poorly documented. With the transfer syntax included in the forthcoming Petri-net standard (see part 2 on "Transfer Syntax" of [5]) this might change in the future. However, it will probably take some time, till the Petri-net Standard is widely adopted and adhered to.

   Meanwhile, there are tools that are designed to be very modular, so that part of their functionality could be used. For instance, the POSES++-Tool has a client responsible for editing and displaying models and a server responsible for executing models. It might be a viable option to produce a new specialised client using the server and thus making use of the excellent server-implementation.

3. *take a big integrated tool and adapt/enhance it*
   For this approach, it is vital that the tool has an open structure and thus allow for enhancements. The relevant information on the structure of tools, interfaces, and internal representations, however, is not easily obtained: for commercial tools, this is often classified information, for non-commercial tools, it is rarely documented in a usable way.

Each of the routes sketched above incurs considerable implementation effort, so that it is not clear, whether the effect can justify the effort. In the following section,

[2] These are the only two I am aware of.

[3] PED is an editor for hierarchical Petri-nets. It exports to INA, PROD, PEP, and TimeNET or in the Postscript, FrameMaker, or XFig-formats. Documentation is currently being worked on.

criteria are developed and the tools available to us are evaluated according to these criteria.

2 The selection process

In the past, there have been several tool surveys already. As a first step, these surveys were consulted. The tool surveys presented at the "Applications and Theory of Petri Nets" in 1986, 1989 and 1992 (see [10, 8, 9], respectively) and the report [7] turned out to be too old to be meaningful to us – a number of very interesting tools have been released in the past few years.

The survey [11] of Heiner and Deussen only considers three tools. It considers the efficiency of the implementation of certain analysis methods and gives benchmark tests and compares the results. The tools considered (INA, PEP and PROD) are interesting mainly for the academic community.

Another study has been conducted very recently by D. Wikarski (see [27]). It compiles short profiles (12 criteria) of 53 tools in the style of the DAIMI tools database. Of these, 16 are presented in more detail by elaborating on the criteria and giving some impressions of using the tool.

The most ample and current source of information on Petri-net tools are the tool databases publicly accessible in the World Wide Web (WWW). There are three big databases at [4, 16, 25]. Together, these sources have been the starting point of this evaluation. All of them have some drawbacks: The tool databases in Milan and Dortmund seem not to be maintained any longer. The database at DAIMI is not complete, in particular older tools do not appear here.[4] Wikarski's study is quite current, and it covers a large number of tools, though in much less detail as one would wish, and without a proper documentation of his approach.

So, a new survey was launched, that builds on existing sources. In particular, the present report is a complement to Wikarski's study with respect to scope and depth.

This survey was conducted in three rounds: Starting from a very large number of tools, most of them were eliminated after some "hygiene" criteria in a first round. Thus focused on a smaller number of tools, these could be evaluated more thoroughly in a second round. In a third round, the tools were classified and compared by a measure defined to reflect our requirements. In order to raise the reliability and quality of this survey, a draft version has been sent to the providers of the evaluated tools, in order to uncover omissions and errors. For the tools Alpha/Sim, Artifex, Design/CPN, Netmate, PEP, PNTalk, and Thorn/DE, feedback has been given by the suppliers.

2.1 The first round: eliminating unfit tools

First, I compiled all names of tools that were mentioned in either of the above sources, if some contact address was provided. This yielded the list shown in table 2. From

[4] In fact, those tools not appearing in the DAIMI database tend to be either abandoned, or of very limited functionality and/or quality.

this list, I eliminated all those tools that were obviously not suitable for the purpose at hand. In particular, all tools were excluded, that (a) did not deal with high-level nets, that (b) did not provide editing, simulation, and analysis functionality, or that (c) require some unavailable preliminaries. In the following, these criteria are elaborated.

Class of Nets All tools are required to deal with high-level nets, i.e. allow for coloured (i.e. model checkingtyped) tokens of some form. As types, all concepts known from programming languages are desirable (and useful), that is, a set of basic types such as enumeration types, bool, real, int, char, string; a set of constructed types such as lists, arrays, tuples, and records; and possibly also parametric and abstract types. Also, all tools are required to provide some form of abstraction mechanism (such as transition or place refinement, transition invocation, subnet composition) and the necessary special net elements (e.g. places shared between nets, port-places etc.). For these formalisms a sound theoretical basis is required. Note that most abstraction mechanisms, irrespective of their syntactical appearance and abstraction philosophy (top-down hierarchic decomposition or bottom-up modular composition) tend to implement the same semantic concept, but fail to have a proper definition, or, if a definition is present, fail to faithfully implement this definition.

Functionality Next, as a minimum of functionality, an integrated package of editor, simulator/animator, and some analysis tools is required. All tools need to have a GUI and be reasonably stable. The functionality of an editor could include facilities for printing, formatting, documentation, and version control. Analysis tools might provide , reachability/coverability analysis, S/T-invariants, statistical analysis, structural properties, and other (model-specific) properties. Simulators can be compared according to their speed and flexibility. Additional functionality might include code generators, alternative front ends (featuring for instance SDL), interfaces to DB- or GUI-tools. Some tools in list 2 are really parts of a package, or are earlier versions of other tools that have been renamed.

Prerequisites All tools have specific hardware or software requirements. Where these could not be met by the environment available at LMU, tools could not be evaluated. Also, some manufacturers did not reply on requests for information on their products or failed to provide evaluation copies. This is the case, for instance, for Bakkenist Management Consultants (ExSpect), for ADV Technologies (ELSIR) and Politecnico di Milano (Cabernet).

Note, that many tools can be eliminated by more than one criterion.

2.2 The second round: testing tools

In the second round, a set of core requirements was developed from the project objectives. These requirements were then translated to a list of criteria, according to which all tools surviving the first round (see table 4) were compared.

The core requirements for the tool to be used in the project have been described in section 1.1. In sections 4.3.2 through 4.5, these criteria are elaborated, and the tools from table 4 are rated according to this catalogue.

The test results have then been sent back to the suppliers of the tools to be checked by them. A number of omissions and some errors were corrected this way, though it is still possible that errors appear in this report. These, of course, are entirely the authors responsibility.

2.3 The third round: classifying and comparing tools

In the third round of the evaluation, a ranking of the most important features of the tools was made by weighing the respective factors. The association of judgements to points was fixed as shown in table 1.

Ratings	++ yes plenty very good	+ (yes) enough good	+- ((yes)) some sufficient	- (no) few bad	-- no none very bad ?
Points	4	3	2	1	0

Table 1: Relation of ratings to points.

Transforming the rating into a score, the tools are compared, classified, and ranked in section 5. Based on this ranking and a discussion of factors not included in the ranking, recommendations concerning the future use of tools in the project sketched above are given. Any features, that have been disabled in the evaluation/demo copies, or that are absent in the current public release version could not be evaluated. Where this is the case, a question mark is inserted, and the feature is rated absent (i.e. 0 points). The same applies for features that could not be evaluated for technical reasons (missing hardware, for instance). Additionally, in some cases it is possible to give an educated guess about the rating. This can be based on the documentation or on the menus available, for instance. In this case, the rating is superscripted by a question mark.

3 Round 1: eliminating unfit tools

The first survey yielded a list of 91 (!) Petri-net tools. In order to manage this huge amount of tools available, I relied on the information provided in the WWW databases and previous surveys being both accurate and current. As for accuracy, this means that no contradicting information was given by the sources used. Although I encountered a number of (printing) errors and incompletenesses, this assumption is indispensable due to the finiteness of the author's resources. As for currency of information, again, the most recent information available in any of the sources was considered. Given that anybody having implemented a (good) tool has a natural interest in making this known, I think this assumption is justified.

ALPHA/Sim	Anarco	Artifex	Cabernet	Capsnet
Chromos	CodeSign	Combag	Comdes	CorMan
CPN-AMI	Design/CPN	DESIGN/OA	Diogenes	DNAnet
DNS	DSPNExpress	EASE	EDS	ELSIR
ExSpect	F-Net	FORSEE	FUN	GRAF
GraphEd	GreatSPN	HiQPN	Hypernet	INA
INCOME	IPTES	Looping	LOOPN	MACROTEC
MERLOT	METADesign	MISS-RdP	Moby	MoPED
Netman	Netmate	NETOBJ	ONE	ORIS
PACE	PAPETRI	PAREDE	PED	PENECA
PEP	PESIM	PETRIMaker	PetriSim	PETSI
PN^3-Editor	PNAnalyser	PNS	PNTBLSIM	PNTalk
POSES++	PROD	Product Net Mach.	PROMPT	PROTEM
PSItool	QPNTool	SANDS	SEA	SPNP
SPN2MGM	STROBOSCOPE	SURF-2	SYROCO	SystemSPECS
TemPRO	Thorn/DE	TimeNET	TORAS	UltraSan
Visual SIMNet	Visual Object Net++	Voltaire	WAM Subset	WebSPN
WinPetri	XPETRI	XSimNet	YANED	

Table 2: All tools considered.

Note, that there is a set of tools covered by Wikarski, that I have been unable to gather enough information about to include them in the third round, most notably ExSpect and ELSIR. Table 3 shows what tool has been excluded for what reason. After this selection, the following tools remained (see table 4). A number of tools have been partially evaluated, but the quality of data obtained so far is not sufficient to have them included in this survey. This is mainly due to technical problems with the installations. This applies for ALPHA/Sim, Looping, Miss-RdP, Poses++, and SEA. I hope to be able to include them in a future version of this survey.

4 Round 2: testing tools

In round 2, the tools in table 4 were analysed in greater detail according to the criteria above. In a first section, general information on technical requirements, prices, and license conditions are compiled. In the subsequent sections, the main components (Editor, Simulator, and Analyzers) are compared. After that, the quality of the user interface and the documentation are surveyed. Finally, some technical aspects are considered. Special attention is paid to the technical aspect of "openness", i.e. the degree to which a tool lends itself to being integrated with or used/enhanced by some other tool.

Class of Nets	
insufficient colouring	Capsnet, Chromos, CorMan, Diogenes, DNAnet, DSPNExpress, EASE, EDS, F-Net, GreatSPN, Hypernet, METADesign, MoPED, NETOBJ, ORIS, PAREDE, PESIM, PETSI, PETRIMaker, PNS, PROD, SPNP, SPN2MGM, STROBOSCOPE, SURF-2, SYROCO, TimeNET, TORAS, WebSPN, UltraSan, Visual Object Net++, WAM Subset, WinPetri, XPETRI
insufficient abstraction	Capsnet, Combag, Comdes, DNAnet, DSPNExpress, EASE, ELSIR, Graspin, GreatSPN, Hypernet, INA, IPTES, Looping, MACROTEC, METADesign, Moby, ORIS, PAPETRI, PetriSim, PETSI, PNAnalyser, PNS, PROD, PROMPT, QPNTool, SPNP, STROBOSCOPE, SURF-2, SYROCO, TemPRO, TimeNET, TORAS, UltraSan, Visual Object Net++, Visual SIMNet, WebSPN, WinPetri, XPETRI
Functionality	
	Anarco, Chromos, Combag, Diogenes, EASE, ELSIR, FUN, GraphEd, Graspin, INA, IPTES, Looping, LOOPN, MACROTEC, Moby, MoPED, Netman, ONE, ORIS, PACE, PENECA, PED, PETSI, PNS, PNTBLSIM, PN^3-Editor, PROD, Product Net Machine, PROMPT, QPNTool, SPNP, SPN2MGM, SURF-2, TORAS, UltraSan, WAM Subset, XSimNet, YANED
Prerequisites	
insufficient information	ELSIR, ExSpect, Looping, Voltaire
technical constraints	CPN-AMI, Diogenes, EASE, GRAF, Graspin, MACROTEC, MISS-RdP, Moby, ONE, PetriLab, PNTBLSIM, POSES++, Product Net Machine, PSI-Tool Net, SANDS, SEA, WAM Subset
parts of a larger tool	DESIGN/OA, FORSEE
precursors	DNS (Thorn/DE), PROTEM (TemPRO), SIMNet (XSimNet)

Table 3: Reasons for excluding tools in the first round.

ALPHA/Sim	Artifex	CodeSign	Design/CPN	HiQPN
INCOME	Netmate	PEP	PNTalk	Thorn/DE

Table 4: Tools to survive round 1 of the selection.

4.1 General information

In this section, some general facts are collected, that do not affect the evaluation and appear only for the purpose of information. Unless trivial, the columns are explained in the text. For easier reference, the **title** of the columns is emphasised in bold style.

4.1.1 Providers, Packages, Prices

All prices given are for educational institutions only and exclude VAT unless otherwise stated. The Windows NT version of Thorn/DE is not freely available. Concerning platforms and Preliminaries, the following abbreviations and remarks are used:

W3.11	Microsoft Windows 3.11.
W95	Microsoft Windows 95
NT	Microsoft Windows NT.
UNIX	various variants of UNIX, including at least Solaris, most of the time also Linux.
PP VW	ParcPlace Visualworks

All tools have been tested in their Solaris-version except for INCOME and Netmate which were tested under Windows 3.11. The column **Language** indicates what language has been used for the documentation and the menus. Other information such as articles and internal or technical reports are not considered. "x/y" means that most information is in language x, but some is in language y. "x+y" means that all information is in language x, but some is also in language "y".

Further technical support, maintenance, and training on a commercial basis are available for Alpha/SIM, Artifex, and INCOME for (substantial) extra fees. The free tools (CodeSign, Design/CPN, HiQPN, MISS-RdP, Netmate, PEP, PNTalk, Thorn/DE) offer varying amounts of free limited (technical) support on a colleague-to-colleague basis.

4.2 Class of Petri-nets

In this section, the class of Petri-nets that a tool processes are compared. This includes the inscription languages for the net elements and the hierarchy constructs provided. While not exactly equivalent in a mathematical sense, the classes of nets supported are basically the same, and certainly from a modellers point of view. A very

Tool	Version	Provider	Platforms + Infrastructure	Academic Price (excl. VAT)
ALPHA/SIM	1.0.10	ALPHATECH INC., Burlington, MA, USA	Solaris, SunOS, NT	US $ 750,-/100,-
Artifex	4.1-0 beta	Artis S.A., Torino, I	various Unix, NT; C/C++, make	US $ 1000,-
CodeSign	1.1	ETH, Zürich, CH	Smalltalk 80 (PP VW 2.5)*	free
Design/CPN	3.0.4	Uni Aarhus, DK	Solaris, Linux, Mac; SML*	free
HiQPN	2.0	Uni Dortmund, D	Solaris	free
INCOME	3.3	Promatis GmbH, Karlsberg, D	NT, 3.11 + Oracle 7*	DM 2800,-
Netmate	2.3	Uni Kaiserslautern, D	3.11, W95, NT	DM 1500,-
PEP	1.6g	Uni Hildesheim, D	Solaris, Linux	free
PNTalk	0.5 beta	Uni Brno, CZ	Smalltalk 80 (PP VW 2.0)*	free
Thorn/DE	2.3	OFFIS, Oldenburg, D	Solaris, SunOS, Linux, NT + C++	free

Table 5: All tools, that were examined more closely. See text for an explanation of the abbreviations. * means: is included in distribution. The NT-version of Thorn/DE is (currently) not publicly available.

instructive though somewhat outdated overview over a large number of formalisms for High-level nets can be found in [13].

All tools considered can deal with some variety of High-level nets (see table 7). Thus, there must be a notion of **Inscription language for Colours**, **Guards**, and **Arcs**. Also, many tools allow associating *actions* to transitions that are executed on/while firing. In the column for arc-inscriptions, "implicit" means, that implicitly, the only inscriptions allowed are single variables. Typically, there is a set of predefined functions (or macros) available for the inscriptions. With INCOME, for example, this is intended to cover almost all inscriptions occurring. In Design/CPN on the other hand, few functions are provided specifically for use as inscriptions. Instead, the full functionality of SML-libraries is available.

Tool	Package provided	Restrictions of use and additional constraints	Language
ALPHA/Sim	SW on tape/disc, 1 set of printed documentation, 1 year of maintenance US $ 750,-: 20 full licenses, US $ 100,-: 1 student license	non-commercial use only, model-size limited to 75 transitions (possible exceptions)	English
Artifex	SW on CD, 20 floating licenses (UNIX only), 1 set of printed documentation, no support	non-commercial use only, customer must provide regular reports on work involving Artifex	English
CodeSign	SW via FTP	non-commercial use only	English
Design/CPN	SW via FTP	usual legal constraints	English
HiQPN	SW via FTP	non-commercial use only	English
INCOME	SW on CD, 6 days of training, no license restriction	non-commercial use only	German
Netmate	SW via FTP	non-commercial use only	German
PEP	SW via FTP	non-commercial use only	German/English
PNTalk	SW via FTP	non-commercial use only	English
Thorn/DE	SW via FTP	usual legal constraints	English+German

Table 6: Packages provided and license conditions.

Tool	Inscription language for			
	Colours	Guards	Actions	Arcs
ALPHA/Sim	records/2-dim. arrays of: •, bool, enumerations/ranges of int, real, string (all menu based)	simple expression language, predefined functions	--	implicit
Artifex	C++			implicit
CodeSign	Smalltalk			implicit
Design/CPN	CPN ML (dialect of SML)			
HiQPN	explicit enumeration of colours only, guards by unfolding		--	implicit
INCOME	$(\mathrm{NF})^2$-DB-Scheme	extensive library of predefined macros, user-defined macros (Prolog?)		implicit
Netmate	--	and, or, xor	boolean variable assignment	implicit
PEP	subsets of $X \cup X \times X$ where $X = \mathbb{N} \cup \{\bullet, true, false\}$	multisets of terms using the usual boolean and arithmetic operators		multisets of tuples of constants and/or variables
PNTalk	Smalltalk, nets, Prolog-like lists	method calls	method calls, variable assignment	multisets of constants, variables, lists
Thorn/DE	predefined types, C++-types (including pointers, `structs`, and classes)	C++		fixed length arrays of variables

Table 7: Inscription-languages supported.

Table 8 below indicates what means of abstraction are provided. It is not possible to go into any detail of hierarchy concepts in Petri-nets here, and thus the classification given below is quite artificial in many respects. A good introduction to some of the more widespread hierarchy-constructs is described in [13, chapter 7]. Note that the top-down approach (stepwise refinement of net elements yielding a hierarchy), and the bottom-up approach (abstracting from and composing subnets yielding a modular structure) irrespective of their different appearance lead to almost the same implemented features in most tools, as refinement is mostly of the substitution-type (aka. macro-expansion). In Thorn/DE, refinement of transitions by invocation is allowed, though, "I/O-places" are admissible, that corrupt this well-founded conception to a certain degree.

The hierarchy constructs available and the abbreviations used in table 8 are as follows:

T-S/s	Transition-Substitution (shared)	A transition is just an abbreviation for a net. Several different transitions that are refined to the same net collectively refer to a single instance of said net, i.e. they sahre this net.
T-S/d	Transition- Substitution (disjoint)	A transition is just an abbreviation for a net. Several different transitions that are refined to the same net each refer to an individual copy of that net, i.e. the refinements are disjoint save for separately defined shared net-elements.
T-I	Transition-Invocation	Every firing event creates a new instance of the refinement net.
S-S/s	Place-Substitution	As T-S/s, but for places.
S-S/d	Place-Replacement	As T-S/d, but for places.
G	Graphical abstraction	A distinct kind of node without underlying semantics is used to abstract from parts of nets. These nodes are purely graphical by nature, and can be understood entirely as a navigational aid.

Trivially, a substitution can also model a replacement by providing so many copies of a refinement net manually. This is not true the other way round. Also, shared places can implement a kind of composition. Note that Artifex for instance supports several completely different and unrelated means of abstraction, while the approach taken in PEP tightly couples a parallel programming language ($B(PN)^2$, which features full-blown procedure-call with call-by-value/reference etc.), a process algebra (the Petri Box Calculus, PBC), parallel finite automata and high-level nets (M-nets).

When using abstraction notations, it is desirable that the graphical representation either has no connotation with respect to its meaning, or that the semantics implied and the one implemented correspond closely. See figure 1 for the abstraction net-elements used in the various tools, and the list below for a short explanation of their meaning.[5] For instance, boxes are traditionally used to depict transitions. Using boxes for refined transitions suggests a transition-like behaviour of the refinement net[6]. Usually, this is not the case, and the symbols chosen defy the actual semantics.

For instance, "hierarchic transitions" and "Out-Ports" in Artifex, or the "hierarchy substitution transitions" in Design/CPN (Items (d), (b), and (g), respectively, in figure 1.) do not at all behave like transitions and places, though they have similar shape.[7]

Where an abstraction mechanism does not match the intuitive meaning of a net element, it is better to use a fresh symbol without intuitive appeal, as for instance the graphical abbreviation of subnets in PEP, or Artifex/CodeSign (see items (j), and (f) of figure 1 respectively), or the uni-/bidirectional Interfaces of CodeSign (items (k) and (l) of the same figure).

Netmate offers place refinement with interpreted nets only. Contrary to the documentation, in Thorn/DE, priority-queues are not implemented.

For a more general study on Petri-net tools, this section has to be much enlarged, including also capabilities to handle stochastic, hybrid, and low-level nets as well as special annotations for tokens (e.g. timing). Note, that some of the features mentioned above can be modelled by others as well. For instance test, flush and inhibitor arcs and queueing places can easily be modelled by small subnets with appropriate guards. Depending on the semantics used, sometimes these substitutes may have a slightly different, but equally valid meaning.

4.3 Functionality

In this section, I shall compare the functionality offered by the tools. As a consequence of the selection process, all tools have graphical editors and interactive simulators of varying quality. Apart form that, almost all tools have some functionality specifically suited to their purpose. See table 9 for an overview of the functionality and the subsequent sections for a detailed comparison.

4.3.1 Integration and convenience

In the first column (**Installation Procedure**) of table 11 below the difficulty of the installation procedure is shown, where an easy installation gets a high score. In the column **Intuitive and consistent GUI** the usual human-computer-interaction features are subsumed. As an example consider supplying and integrating several ways of achieving the same functionality, such as key-short cuts, menus, and context-sensitive

[5]Note, that typically several special net elements are used together to model an abstraction. For instance, transition refinement usually goes along with interface-type places.

[6]For instance atomic terminating switching without memory, side effects or internal divergence or deadlocking. See [24, pp. 65–77] for a detailed analysis of the intuition of a transition.

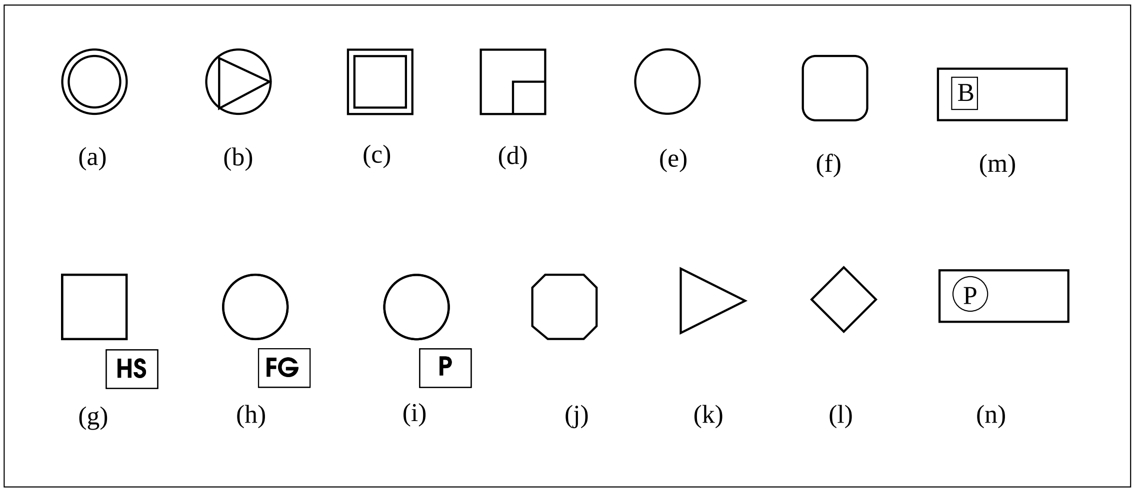
[7]The refined transitions are just a macro-type abbreviation, the place does not hold tokens, and cannot be in the preset of a transition.

Tool	Composition Borders	Refine. Type	Aux. Means for Abstraction	Special Modelling Places	Transitions	Arcs
ALPHA/Sim	Arcs	G[1]	--	LIFO, FIFO, priority queue	constant/ stochastic duration	probability, priority, test, inhibitor
Artifex	Places	T-S/s, G	in, out, and shared places	queues	constant/ stochastic delay/ duration, priority	--
CodeSign	Arcs	G	-	capacities	--	--
Design/CPN	--	T-S/d	in, out, in/out, and shared paces		delay, duration	--
HiQPN	(shared S-S)	S-S, S-R[2]	refined	various queues, capacities	priority, probability, time	weight
INCOME	--	T-S/s[?]	--	organisational units, fixed cost/earning, fixed delay/ duration/ others, actions LIFO, FIFO, priority-queue, capacity	priority	probability
Netmate	Arcs (shared variables)	S-S/d	shared places	capacity	--	marking-dependent weight, time
PEP	Transitions,Places	T-S/d, G	shared places	capacity	--	weight
PNTalk	--	T-I	shared places	--	duration	--
Thorn/DE	Places	T-I, T-S/d	in/out, and shared places, stop-transitions	LIFO, FIFO, capacity	constant/ stochastic delay/ duration, (continuous), capacity	inhibitor, test, flush, weighted, (continuous)

Table 8: Means of abstraction supported. See text for further explanation of abbreviations. A parenthesised name indicates the availability of this feature through another tool. [1] means: boxes play a role in analysis for grouping statistics. [2] means that the refinement nets are of a restricted form only.

Tool	Analysis		Interface to		special or distinctive features	intended/main area of application
	quant.	qual.	DB	GUI		
ALPHA/Sim	yes	no	(no)	(no)	easy to use	m.s.a. of discrete event systems
Artifex	yes	no	yes	yes	generation of distributed code, GUI	m.s.a. of discrete event systems, development of C++-application
CodeSign	yes	no	no	no		m.s.a. of production systems
Design/CPN	yes	yes	via SML		provides access to full SML interactive compiler	m.s.a. of systems/research
HiQPN	yes	yes	no	no	Analysis can exploit hierarchical structure	scientific research
INCOME	yes	no	yes	no	data and organisational models (NF^2, organisation chart)	m.s.a. of workflows
Netmate	no	yes	no	no	special modelling constructs & code generation for PLC	m.s.a. and analysis of automatic control systems (PLCs), education
PEP	no	yes	no	no	Generation of nets form Process Algebra (PBC), Parallel Programming Language ($B(PN^2)$) or parallel finite Automata (PFA)	verification of parallel programs, testbed for PN analysis algorithms
PNTalk	yes	no	via Smalltalk		object-oriented nets	scientific research
Thorn/DE	yes	(yes)	yes	no	fast distributed and sequential simulation, hybrid nets	practical applications/research

Table 9: Overview of the functionality provided. ”m.s.a.” is short for ”modelling, simulation, and analysis”.



(a)	in-Places (Artifex)	(b)	out-Places (Artifex)
(c)	subsystem (Artifex)	(d)	refined transition (Artifex)
(e)	shared place (Artifex)	(f)	subsystem (Artifex, CodeSign)
(g)	refined transition (Design/CPN)	(h)	shared place (Design/CPN)
(i)	port-place (Design/CPN)	(j)	abbreviated net (PEP)
(k)	unidirectional link (CodeSign)	(l)	bidirectional link (CodeSign)
(m)	unidir. link from box (ALPHA/Sim)	(n)	unidir. link from place (ALPHA/Sim)

Figure 1: Some non-standard notations used by tools in net graphs.

pop-up menus. As far as these features are concerned, Artifex[8] and ALPHA/Sim[9] are ahead of most other tools. A new version of PEP is said to have been much improved in this respect.

One might argue that an abundance of features makes a tool hard to learn, so that a areful restriction of features is in fact a pro. However, there is a difference between simple lack of features, a large number of features, and careful selection and organisation of (few) features. As an example, consider the editors of ALPHA/Sim, Design/CPN, and INCOME. While the ALPHA/Sim-editor simply has few features (no grid, no alignment, no scaling, no colour/font customization), the editor of Design/CPN offers a vast number of scarcely integrated features. On the other hand, the INCOME-editor offers a smaller, yet more powerful and better organised set of features.

The aspects **stability** and **speed** should be obvious. See table 10 for data on the environments used to conduct the tests. By **Integration** I mean the integration of

[8]Though even here the interaction is not always consistent. One example is the availability of several ways to activate a function. For instance, the CR-key should be equivalent to clicking a check-button. In Artifex, this is sometimes the case, though not always. Most other tools do not offer such a feature at all.

[9]ALPHA/Sim has very few context-sensitive (pop-up) menues, does a sloppy screen redrawing, and does not easily allow to deselect arcs when drawing.

Tool	System	Main Memory (MB)	Operating System	Reason for not using Sparc/Solaris
ALPHA/Sim	Pentium-S/100Mhz	32	Windows NT	compatibility problems
Artifex	SPARCstation 10	48	Solaris 2.6	–
CodeSign	SPARCstation 10	48	Solaris 2.6	–
Design/CPN	SPARCstation 10	48	Solaris 2.6	–
HiQPN	SPARCstation 10	48	Solaris 2.6	–
INCOME	Pentium-S/100Mhz	32	Windows 3.11	not available for Solaris
Netmate	Pentium-S/100Mhz	32	Windows 3.11	not available for Solaris
PEP	SPARCstation 10	48	Solaris 2.6	–
PNTalk	SPARCstation 10	48	Solaris 2.6	–
Thorn/DE	SPARCstation 10	48	Solaris 2.6	–

Table 10: Environments used to conduct tests.

Tool	Installation Procedure	Intuitive & Consistent GUI	Stability & Compatibility	Overall Speed	Integration	Auto-Save
ALPHA/Sim	++	+	–	–	++	––
Artifex	++	+	+	+–	+	––
CodeSign	++	+	+	+–	++	––
Design/CPN	++	+–	++	–	++	––
HiQPN	++	–	+	+	++	––
INCOME	+	++	+	–*	++	––
Netmate	++	+	–	++	++	––
PEP	+	+–	+–	–	++	––
PNTalk	++	+–	+	–	+	––
Thorn/DE	++	++	++	++	++	––
weight	0	2	2	1	2	2

Table 11: Integration and convenience of tools. *: according to Promatis, it should be much faster than the tests showed, so that this figure might be the result of some technical problem on behalf of the author.

the tool for the various tasks (editing, simulating, analysing). Note, that integration and openness (see section 4.5) are sometimes difficult, though not impossible to combine.

4.3.2 Graphical Editor

The graphical editor is easily the one component of an integrated Petri net tool, that users are likely to spend the most time using. Apart from any modelling activities which obviously rely on the editor very much, the validation and verification phases make use of the model as it is produced with the editor. So, to a certain degree, validation of a model depends on the editor as well.[10] Hence, particular effort has been put into identifying and evaluating all relevant aspects of Petri-net editors (see tables 13 and 12).

In the column **Documentation Support**, all means to document (large) models are summarised: automatic generation of reports from the documentation information specified with the file, various formats exported and so on. The aspect **Layout Support** sums up all features that support the creation of aesthetically more appealing graphics, including automatic layout. There are two features present in almost all tools:

- **grid**
  A drawing grid allows the proper (automatic) alignment of net elements. This typically comprises the grid as such, an optional "snap to grid" function, and a cleanup-function.
- **alignment**
  Failing to provide a drawing grid, some functions to align net elements are required. Design/CPN, for instance, offers all sorts of horizontal, vertical, and relative alignment with various orientations.

Other graphical elements subsumed as layout support are extra text in net graphs (such as headings), extra graphical elements, special diagrams (such as bar and line charts), other facets of graphical elements (colour, style and width of lines, font, style, colour, and size of text etc.). In Design/CPN, all of these (and more) can be configured. Additional features might include aids for automatic layout support. It is well known, that automatic graph layout is a difficult open problem, but at least some functionality other than simple alignment might be provided, such as smoothing an arc made up of several line segments into a curve (Design/CPN does so).

The aspect **Syntax Support** includes all features, that facilitate the editing of nets and inscriptions. One particularly helpful functionality is the provision of (context sensitive) default values. Also, the built-in support for certain routine-sequences of steps or a macro-facility are very useful. As an example, consider the entering of a new net. Entering a transition might result in the automatic activation of pop-up menus to insert the corresponding inscriptions, or the incident arcs, or another transition. In this respect, ALPHA/Sim is particularly poor, in that it does offer neither a drawing grid nor alignment functions.

The column **Appearance Customisation** judges the number and usefulness of options to customise the graphical appearance of the tool and the models, either as

[10] Consider a user unfamiliar with Petri-nets (or any other formalism) who will find it much more easy to understand a simulation run with intuitive icons instead of just boxes and circles. Obviously, all this applies for graphical editors only.

driven by requirements of the environment (e.g. colour/bw-screen) or by the individual preferences of the user (for instance (colour/shape of net elements, font/size/type of texts).

The column **Printing** includes the quality and convenience of printing output (formats, scaling, output redirection, printing parts/all of a model etc.).

By syntax in column **Syntax Checks**, I mean the net graph, the names of net elements and colour sets, and annotations such as time and colour. There are four levels of checking conceivable:

- **explicit check**
  The least one would expect is a command to completely check the syntactical correctness of a model, and in case this is not so, supports debugging by providing the user with meaningful error messages. Preferably, this type of check is implemented as a separate command, though an implicit check during code generation (if applicable) is also acceptable.
- **interactive warnings**
  Preferably, however, one would like an editor to always immediately mark any inconsistent parts of a model, as it is the case for some attributes for Artifex.
- **default values**
  Many attributes have standard values. For instance, in P/T-nets, any arc bearing no weight is assumed to have weight 1. Analogously, in a coloured net one would expect any uncoloured place to contain standard P/T-type tokens, and any transition without a switching time would be assumed to either consume no time at all, or some unitary amount of time. Providing the user with reasonable default values is a very valuable asset when modelling.
- **restricted interaction**
  Some tools go even further in that they explicitly and actively disallow syntactically faulty entries or inconsistent states. In some cases, this enforces unnatural interaction modes upon the user and is thus a bad thing.

Most tools mix these possibilities. For instance, all tools considered enforce a well structured net graph by disallowing the connection of places and transitions by arcs, though there are notable exceptions to this rule. Artifex and INCOME provide default place colours, Artifex also provides default guards for some simple cases, while Design/CPN requires the user to enter all annotations, no matter how trivial they might be. Artifex issues warnings (a red cross through a place), when a type is associated with the place that has not yet been defined. On the other hand, Artifex forbids inconsistent namings of net elements and colour sets, which can be quite bugging at times. Design/CPN provides a command to check the rest of the syntax.

It should be clear that the last three items mentioned in the previous paragraph (warnings, defaults, restrictions) are integral aspects of an editor. Explicit syntactical checks might also be considered as some simple form of analysis functionality ([27] does so, for instance). In this survey, however, analysis is understood to be more than just syntactical analysis of a model, and thus, syntax checks are included as part of the editing functionality.

In column **Error Msgs.** the facilities for error messages are judged by their amount and helpfulness. They can take the form of graphical modifications. For instance, Artifex will mark inconsistent parts with a red cross.

In column **Version Control**, provision for versioning of net models is considered. This feature is quite important when modelling large systems, as here typically a sequence of versions of a large number of components is produced. Thus, a tool to support the management of several versions is highly desirable. Such a facility is not present, however, in any of the tools except CodeSign, which supports this to a small degree. I some other tools such a facility could be added with very little extra effort. The way to do this is to modify the file format in which net models are saved to permanent storage such that UNIX-tools like `rcs` or `sccs` could be used on these files: This requires nothing more than allowing comments in the first couple of lines. Thus, one would have a "poor mans version control system". The file formats, however, are either undocumented (e.g. Artifex and most other tools), or explicitly disallow comments in the appropriate places (PEP). Additionally, interference with the file structure of tools is not really advisable in general, so that might be necessary to add some support to allow for standard revision control systems to be used.

For large models, hiding of an unnecessary amount of detail and effective navigation in a model is a crucial aid to users (see column **Navigation** in table 13). In Design/CPN, for instance, there are two ways to navigate in a model: either, one can use the **Hierarchy Page** which graphically displays the refinement (or composition) relation, or one can just open a refinement net by double-clicking on the refined transition. Navigation in CodeSign is very similar. In PEP, most annotations can be shown or hidden collectively. CodeSign, INCOME and Artifex always hide annotations, and make them accessible via extra menus. Obviously, scaling can be used as a navigational aid, too.

Also, it is very useful, when inscriptions can be selectively hidden, that is, any *subset* of them, as chosen by the user (see column **Selective Display**). In Design/CPN, it is only possible to hide all inscriptions of a set of places and transitions or a set of arcs altogether, but not some kind of inscriptions (or even all inscriptions) of a complete net. In INCOME and ARTIFEX most attributes can be viewed only via special menus. So, in order to access all details of a model, all net elements would have to be visited and the respective data inspected. In Design/CPN, on the other hand, where it is possible to have all data displayed in a model, it is not always very easy to see what annotations belong to what net element – a particular command had to be added to show just this connection.

4.3.3 Simulator

For many applications of Petri nets, a simulator is an indispensable tool. This aspect is surveyed in table 14. In the first five columns, a number of features is evaluated that allow for convenient simulation. This includes the general **simulation modes**, the availability of **break points** and **watch points** with flexible conditions (at certain times, upon firing/activation of certain transitions, upon certain markings, and so

Tool	Support for Documentation	Support for Layout	Support for Syntax	Appearance Customization	Printing	Syntax Checks	Error Messages	Version Control
ALPHA/Sim	+	−−	++	−−	+−	++	+	−−
Artifex	++	+	+	+−	+	+	+	−−
CodeSign	−−	−−	−−	−−	+	−	−−	+
Design/CPN	+−	+	−	+	+−	+	+−	−−
HiQPN	+	−	−	−−	−−	−−	−	−−
INCOME	++	−	++	++	++	+−	+−	−−
Netmate	+−	+	+	−	?	+	+	−−
PEP	−	+−	+−	+	+	++	+−	−−
PNTalk	−−	−−	+−	−−	−−	−	−−	−−
Thorn/DE	+−	+	++	+	+−	+	+−	−−
weight	1	2	2	1	2	3	3	2

Table 12: General aspects of the editor.

Tool	Navigation in		Navigation by		
	Editor	Simulator	Hierarchy Page	Scaling	Selective Display
ALPHA/Sim	++		++	−−	+−
Artifex	−	++	−−	+	−
CodeSign	++		+		+
Design/CPN	+		+	+	+
HiQPN	+		+	+	+−
INCOME	++		−−	+	+−
Netmate	+		−−	+	+
PEP	+−		++	+	++
PNTalk	++		−−	−−	−−
Thorn/DE	++		+	++	+
weight	3	3	0	0	0

Table 13: Navigation in Models.

on), stepping modes (backward, forward, substeps[11], single, continuous), simulation with respect to the simulation level[12], batch and/or interactive runs, and so on. Note, that combining batch runs with watch points results in traces, that might be exported in a format amenable to further analysis (by another tool).

Many tools compile nets into code in some ordinary programming language (most frequently used: C/C++). In most tools this is done mainly for efficiency reasons. Some tools, however, allow to generate stand-alone applications that can be used as such. This opens the road to conceiving and designing an application or a complete system as a Petri-net, and directly transform it into an application. Thus, the abstraction level in conventional system design could be raised considerably. There are some first traces of this present in the Artifex-tool. In the column **Code Generation**, the languages in which code can be produced are described. Also, it is stated whether the code can be executed locally only or in distributed environments, too.

The column **Speed of Simulation** indicates only superficial impressions, as proper benchmarking is beyond the scope of this survey (see [11] for a first attempt at benchmarking Petri-net tools). Additionally, benchmarking is not possible at all for INCOME, since the evaluation copy does not allow the creation of new simulation runs. One also has to keep in mind that for qualitative modelling with simple animation, speed is not very important, and indeed must not be too fast.

In the column **Animation** the facilities for generating animations are rated. It is helpful to distinguish between **simple** animations that are basically showing a token game and **advanced** animations, that allow for directly coupling GUIs (X Windows, VRML, ...) to net elements. While the latter can be much more informative, the amount necessary to produce them is typically very large. In particular this applies for large models with quantitative bias, where statistical data about token distribution is much more useful than precise information about complete runs. On the other hand, for small models or qualitative models, individual information is indeed necessary. Depending on the kind of users that are expected to be presented with models, different tools provide varying amounts of support for the different types of animation. For instance, INCOME allows simple animation only. By replacing net elements and tokens by icons, a very effective animation can be produced, however.

Almost all tools provide some form of simple animation, though to a varying degree. Due to the effort required, advanced animation has not been evaluated. Again, if no animation is available with a tool, this is indicated by the lowest rating.

4.3.4 Analysis tools

After the editor and the simulator, tools for the analysis of models are the third big kind of tools necessary. The aspects surveyed in this section (see table 15) are mostly self-explanatory, save the abbreviations: The first four columns (**R., L., F., M.C**) stand for Reachability, Liveness, Fairness, and Model Checking. In the **model-checking** column the (temporal) logics supported are indicated, when avail-

[11]In Artifex, withdrawal and emission of tokens can be simulated as separate steps.

[12]Given a hierarchically structured net, one may wish to simulate only a certain abstraction level or skip whole subnets.

Tool	Simulation-modes			Watch	Break	Code	Speed	Animation	
	batch	interactive	rating	Points	Points	Generation	of Sim.	simple	adv
ALPHA/Sim	yes	single step, continuous, interval	+	yes	yes	--	++	--	no
Artifex	yes	single step, continuous, interval	+	yes	yes	located/distr. C++	++	-	yes
CodeSign	yes (forward only)	back/forth single step/ continuous/ fast	++	no	yes	--	+	+	no
Design/CPN	yes	single step, continuous	+	yes	no	-	-	+	yes
HiQPN	no	single step, continuous	+	no	no	--	+	+-	no
INCOME	yes	single step, continuous	+	yes	no?	--	-*	++	no
Netmate	no	back/forth single step	-	no	no	AWL (code for Siemens S5 PLC)?	+	+-	no
PEP	(yes)	single step, continuous	+	no	no	--	+-	+	no
PNTalk	yes	single step, continuous	+	no	yes	Smalltalk	+	--	no
Thorn/DE	yes	single step, continuous	++	yes	no	located/distr. C++?	++	--	no
weight	2	0	3	2	1	0	2	4	0

Table 14: Simulator features.

able/applicable. Without proper benchmarks, the information given here is restricted to binary information (available/not available). The comparisons include efficiency of tools only in a very limited sense. All of these analysis tools deserve a separate survey of their own. Future surveys will have to elaborate this section with quantitative analysis of efficiency and convenience of these features. The **R.P** column rates the quality of result presentation, for instance, whether reachability graphs can be manipulated and printed and so on. In the last two columns, type and usefulness of any other analysis techniques are shown.

4.4 Maintenance, Documentation, and future plans

Documentation (see table 17) is judged by the amount and quality of documentation provided, by the structuring of the texts etc. Tool maintenance consists of support, i.e. technical information via phone and/or email, and what rate and quality of future development can be expected. In particular, enhancements of functionality and usability are given high ratings. The assessment of likely future development (column "**Future Dev.**") is bound to be but a subjective educated guess, based on my personal impressions and the information as issued by the tool providers. By adding a column indicating the **confidence level**, the subjectivity can be reduced. The confidence level expresses the likelihood of the assessment of the prospects of future development given here being accurate. It is based on the amount and type of information that has been used in the assessment. In this column, the following ratings are used:

Rating	Meaning
++	I know the relevant people personally and trust their statements.
+	I have personally met technical staff and/or had frequent mails/calls with them.
+-	Only contact with sales/marketing people, and/or little contact with technical staff.
-	No personal contact, just written material such as advertisement leaflets or README-files.
--	No information whatsoever.

In the following paragraphs, I shall provide additional information justifying the assessment for the various tools individually. Artifex and INCOME apparently will be constantly developed further. One indication of whether a tool will be maintained is the number of installations (see table 6 on this issue). If this number is large, it seems safe to infer a high chance of future extensions to a tool. This is not the only indicator, of course, and those (commercial) tools with a much smaller basis of installed copies are less restricted in their future development (for instance CodeSign)by having to maintain upward compatibility.

The PEP-tool is today a purely academic tool. However, as there are at least

Tool	Dynamic Properties R.	L.	F.	M.C.	Invariants	Statistics	Result Presentation	Others type/rating	
ALPHA/Sim	no	no	no	no	no	++	+−	bookkeeping	−
Artifex	no	no	no	no	no	++	+	bookkeeping	−
CodeSign	no	no	no	no	no	+	−−	none	−−
Design/CPN	yes	yes	yes	(TORAS)?, ML functions on RG	(PROMPT)?	+−	+	home-states, boundedness, bookkeeping	++
HiQPN	yes	no	no	no	(P/T-nets)	+−	−	deadlock/trap	+−
INCOME	no	no	no	no	no	?	++	bookkeeping information about net structure	−
Netmate	no	(INA)	(SPIN)?	no	(INA)?	−−	?	conflicts, some structural properties, timing analysis	−
PEP	yes	yes	no	CTL, LTL	(INA, PROD)?	−−	−−	reduction (PROD)?	++
PNTalk	no	no	no	no	no	−	−−	none	−−
Thorn/DE	(yes)	no	no	no	no	++	+	none	−−
weight	3	2	1	2	1	3	2	0	2

Table 15: Qualitative analysis methods: R. (reachability), L. (liveness of transitions/binding elements), F. (fairness), M.C. (model checking of temporal logic formulas), R.P. (result Presentation". A parenthesised name indicates the availability of this feature through another tool.

three independent groups involved (at TU Munich, Uni Hildesheim[13], and Uni Oldenburg), and considering the quality PEP has reached already, assuming a long-term development and use of PEP seems to be reasonable.

The Design/CPN tool has been discontinued as a commercial tool. Responsibility for further development is now mainly with Aarhus University, who integrate new features into the tool according to their own needs. Probably, other academic groups are working on similar things, too. The Voltaire seems to have been abandoned altogether. Thorn/DE has definitely been discontinued.

Tutorials, examples, and documentation on the user-interface are important aids in using a tool. Both quality and quantity are assessed here. Note that we distinguish between user-interface documentation and technical documentation (see the table of section 4.5 for the latter).

Thorn/DE comes with a number of examples, but without tutorial or sufficient documentation. The tutorial of CodeSign is easy to follow, but covers only part of the functionality. INCOME has such intuitive handling, that even lacking a proper tutorial or additional documentation, the examples provided can be used. Note, that Promatis also provide four days of training and a technical hotline with the tool. PEP provides no tutorial, little documentation, but a large set of meaningful examples.

Some of the tool manufacturers provided interesting information on the next steps of development (see table 16).

4.5 Openness and interfaces

With regard to the aim of using an existing tool for the special purposes laid out in the introduction, it is immediately obvious that one way or the other enhancements will have to be made. These aspects are addressed in this section, see table 18.

The availability of interfaces for **importing** and **exporting** semantic information (such as net graphs, traces, reachability graphs in any well-documented format[14], but not printing and documentation formats) and the openness of a tool are the most important properties. By openness, I mean the effort one has to invest in enhancing and modifying a tool such that it fits the requirements. A tool is considered open to the degree, that the architecture at the module level along with precise interface specifications are known. Given that extensive analysis of source code is not a viable option, and an application **programming interface** (API) in its own right does not exist, it is mainly the tightness of coupling between several components that determines the openness of a tool.

Typically, if a tool consists of components that are only loosely coupled, then the interfaces are easier to exploit for adding other components, though certain restrictions of what functionality of the tool can be accessed (i.e. the degree of integration of the new modules) have to be faced. In general, the coupling of components can be achieved by various methods characterised by the means of communication between components (see table below).

[13]The CS department of Hildesheim University has been closed. Thus, the contribution to the further development of PEP will carry on for a limited time only.

[14]See [1] for instance.

Tool	Future plans
ALPHA/Sim	enhancing the interface to facilitate working with submodels; Windows 95/98 version; improved interface to external code; improved statistics specification and collection.
Artifex	Visual Basic GUIs can be integrated, native Visual C++ support is to be added soon.
CodeSign	Generation of C, VHDL, and Java is planned, but not yet implemented. Better support for statistical analysis of simulation results.
Design/CPN	Redesign of simulator, syntax driven editor for declarations with incremental syntax check and code generation, invariant tool, place capacities, FIFO places, inhibitor/test arcs, synchronised transitions, condensed state spaces, textual interchange format.
HiQPN	Further improvements of analysis tools, improvements to GUI.
Netmate	Support for coloured Petri-nets, C-code generator,
PEP	Integration of more import-/export-interfaces, VRML-interface, new analysis algorithms, improvements to GUI. Version 1.7c is to be released shortly which is said to be much improved.
PNTalk	Transitions with delay, support for printing and documenting nets, documentation, qualitative and quantitative analysis, reimplementation of tool.
Thorn/DE	Project has been abandoned at original site, but work on Thorn/DE will continue at Stuttgart University.

Table 16: Planned development steps of the tools

Tool	Support	Future Dev.	Confidence Level	Tutorial	Examples	User Docu.	Help Function
ALPHA/Sim	+	+	+−	+−	++	++	−−
Artifex	++	+	+−	++	++	++	+
CodeSign	−	+	++	+	++	+	−−
Design/CPN	+	++	+	++	++	++	−−
HiQPN	−	−	+−	−−	++	++	++
INCOME	+	+	+	+	++	+	++
Netmate	−	+−	+−	+	+−	++	++
PEP	+−	++	++	−	++	+−	+
PNTalk	−−	++	−	−−	++	−	−−
Thorn/DE	−	−−	+	−−	++	++	−−
weight	0	1	0	2	2	4	1

Table 17: Prospective future development and user documentation.

.

communication via	coupling	openness
memory/data structures	tight	small
database/repository	medium	medium
text-interface (commandline, TCP/IP)	loose	large

Apart from availability of an API and the type of coupling, technical documentation (such as the specification of file formats, DB-schemes, and data structures) also contributes to the openness of a tool. INCOME is said to be an open system[15], though no evidence of this claim has been produced.

Tool	Formats (type/rating)		Technical	Programming Interface	
	exported	imported	Docu.	Type	Rating
ALPHA/Sim	Net graph*/+-		+-	user defined functions can be integrated only recompiling	-
Artifex	Traces/-	--/--	++	C/C++/Visual Basic call, OLE	+
CodeSign	Net graph*/+-		-	Smalltalk-call	+
Design/CPN	(HACOC: ASCII)/+		++	SML-call	+
HiQPN	Traces, APNN/+		--	none	--
INCOME	Net graph*/+-		--	SAP R/3-RFC, Oracle	-
Netmate	INA/+	--/--	--	none	--
PEP	PROD, INA, ASCII, SMV, Spin/ ++		+	Command-line interface, hooks	++
PNTalk	Net graph*/+-		--	Smalltalk-call	+
Thorn/DE	Net graph*/+-		+-	C++-call (NT-version: OLE)	+
weight	7	7	6	0	7

Table 18: Features contributing to openness of a tool. * means: undocumented ASCII format.

5 Round 3: comparing and classifying tools

In this section, the data assembled in the previous sections is weighted and interpreted. Based on this discussion, a classification of tools is proposed, and recommendations for the future use of tools are made.

[15]In a letter, Promatis claim that "INCOME ist ein offenes System (Darlegung aller Codes)" i.e. "INCOME is an open system (all codes are laid open)".

5.1 Usage profile for relative rating

The relative factors of the various criteria that have been shown already in the tables in the previous section, are shown again synoptically in table 19. This table can be seen as a profile of the expected usage. Following this measure, the analysis of the preceeding sections results in the following amounts of points, and the subsequent ranking (see tables 20 and 21).

By providing multiplicity factors in the comparisons of the previous sections, the evaluation scheme used here has been made clear. In particular, this way it is very obvious, what aspects are considered in the ranking given above. A decision in favour or against any of the tools should also include all unquantified factors, and "soft facts" derived from the future use of a tool. For instance, if a tool is to be used in classroom teaching, more emphasis must be put on stability, learnability, and the availability of tutorials. Obviously, in that case the price per license also becomes an important argument then. If the tool is to be used for a long time, good development prospects might outweigh insufficient functionality. These points, however, vary for every concrete situation, and can thus not generally be discussed here.

It should be stressed again that the evaluation and the ranking are independent of each other. The purpose of this is to be able to provide different profiles and to automatically obtain a ranking such as shown in table 21 for the profile given in table 19. Interest in developing other profiles like this has been expressed in the Petri-net community already.

5.2 Classification of tools

Unsurprisingly, there is no obvious "best tool" – all tools have their particular strengths and weaknesses with respect to certain applications. Thus, for a more general survey, it would seem more adequate to give a classification of tools according to their intended application area rather than just a simple ranking. In general, it seems as if the tools considered here can be broadly classified as either mainly for academic or mainly for industrial purposes, according to the tasks they are optimised for. Usability for either of the two appear to be almost mutually exclusive in the tools considered here.

The main tasks a Petri-net tool is supposed to perform can be summarised as modelling, validation (by simulation), execution, and analysis (quantitative and qualitative). Industrial applications tend to have a descending priority in this spectrum, while it is the other way round for scientific applications:

Relative Priority for Industrial Applications	Task	Relative Priority for Scientific Applications
high	modelling	low
	validation	
	execution	
	quantitative analysis	
low	qualitative analysis	high

Group	Aspect	Factor	Max
Editor	Support for		
	Documentation	1	
	Layout	2	
	Syntax	2	
	Syntax Checks	3	
	Version Control	2	
	Navigation (Editor)	3	52
Simulation	Modes		
	Batch	2	
	Interactive	3	
	Watch Points	2	
	Break Points	1	
	Speed	2	
	Animation	4	
	Navigation (Simulator)	3	68
Analysis	Reachability	3	
	Liveness	2	
	Fairness	1	
	Modelchecking	2	
	Invariants	1	
	Statistics	3	
	other analysis methods	2	
	Result Presentation	2	64
Tool	Appearance Customisation	1	
	Printing	2	
	Error Messages	3	
	GUI quality	2	
	Stability & Compatibility	2	
	Overall Speed	1	
	Integration	2	
	Auto-Save	2	
	Future Development	1	64
Documentation	User	4	
	Technical	6	
	Tutorial	2	
	Examples	2	
	Help Function	1	60
Openness	Programming Interface	7	
	Import	7	
	Export	7	84
	Max Points		392

Table 19: The relative weight of the factors.

Tool	Points						Sum	Rank
	Editor	Simulation	Analysis	Tool	Docu.	Openness		
ALPHA/Sim	35	49	18	33	40	35	210	5
Artifex	28	53	20	49	59	28	237	4
CodeSign	21	54	9	31	32	35	182	7
Design/CPN	28	46	56	38	56	63	291	1
HiQPN	16	35	27	23	28	42	171	8
INCOME	32	55	10	44	30	21	202	6
Netmate	32	26	10	32	30	21	131	10
PEP	27	37	36	36	39	84	259	2
PNTalk	19	39	3	21	12	49	143	9
Thorn/DE	37	54	27	41	36	49	244	3
max	52	68	64	56	60	84	392	
best	37	55	56	49	59	84	291	

Table 20: Applying the measure to the tools considered.

Free	Rank	Tool	Points
√	1	Design/CPN	291
√	2	PEP	259
√	3	Thorn/DE	244
	4	Artifex	237
	5	ALPHA/Sim	210
	6	INCOME	202
√	7	CodeSign	182
√	8	HiQPN	171
√	9	PNTalk	143
√	10	Netmate	131

Table 21: Ranking according to the scheme from table 19.

This classification is of course only a very coarse grid and the ranking given in the previous section is a specific measure for the requirements of the specific project mentioned in section 1.1. Other projects will have other measures, and thus result in a different ranking, and even require different and/or elaborated evaluation schemes: it is beyond the scope of this survey to define detailed profiles for, say, industrial applications, classroom use, or workflow management, and base a generalised classification on them. It should be clear, however, how this could be done in a way similar to the profile presented in table 19.

Apart from the data collected above, there are other useful indicators helping to decide in favor of or against using a certain tool. In particular, prior experience of other users with a tool can be very helpful. In table 22, some initial data is collected on how long a tool has been in use, and how many licenses have been issued. Note, that some tools issue only site licenses, while others issue both individual and site licenses, so that the number of licenses may be much smaller than the number of actual users. Most tool suppliers will also be glad to provide more detailed information or reference customers. Contact addresses and emails can be found on the respective WWW-sites.

In table 22, the figures concerning numbers of places and transitions are mostly estimates by the tool-providers. They allow very little assertions about the size of state spaces a tool is capable of handling: with infinite token types (i.e. colour sets), it is not possible to unfold a high-level net at all. Also, apart from the size, the structure of the state space is also an importatnt factor: a large symmetric state space can be easier to analyse than a small but complicated state space.

Most tool suppliers saw not fit to supply meaningful data on the size of state spaces[16], so these data are not included here. If no qualitative analysis is intended anyway, the size is also immaterial.

Tool	in use since	Number of Licenses issued?		largest high-level model	
		academic	industrial	Places	Transitions
ALPHA/Sim	9/94	8	22	2292	1849
Artifex	1989	n.a.	n.a.	1321	883
CodeSign	4/97	71	64	130	150
Design/CPN	1989	286	103	2000	300
HiQPN	n.a.	n.a.	n.a.	n.a.	n.a.
INCOME	n.a.	n.a.	≥ 39	n.a.	n.a.
Netmate	n.a.	n.a.	n.a.	n.a.	n.a.
PEP	11/1995	>500		10^5	10^5
PNTalk	prototype only	35	7*	1023	744
Thorn/DE	9/1995	~30	5*	4900	2300

Table 22: Actual applications of tools. Further explanations in text. * means: evaluation copies. "n.a." stands for "not available"

To conclude this comparison, I feel it is necessary to repeat, that this evaluation is relative, i.e. the number of points given for some feature of some tool makes sense only in relation to the corresponding feature of the other tools. All of the tools selected in round one are high-end tools.

[16]The notable exception is Artis, who figured that unfolding the largest Artifex-model yields a CE-net with 29484 places and 28398 transitions.

5.3 Further information

Depending on the intended area of application, there are many other issues one might raise in connection with comparing tools. Also, the information presented here are not detailed enough to answer some questions on the tools considered. Furthermore, the information presented here is already slightly outdated by the date of publication. The reader interested in finding out more about Petri-net tool will have to refer to the providers of the various tools, or the other publicly available sources, in particular the WWW.

A good starting point for gathering further information on specific tools are the WWW -pages of DAIMI at [4]. There, one also finds the names and emails of persons to be contacted in order to obtain more detailed information, in particular about any changes since this survey has been conducted, evaluation copies, or examples of models for the various tools.

5.4 Recommendations

One of the most surprising facts was the quality of the academic tools as compared to the commercial products: though the commercial tools offered better user interfaces, some of the academic tools are more than sufficient for real world applications in terms of usability. Unsurprisingly, concerning qualitative analysis and openness, the commercial tools are no match for the high-end academic tools. Thus, for the purpose at hand, I can not recommend purchasing any of the commercial tools at this time. Given the different strengths and weaknesses, I recommend using Design/CPN and PEP.

In particular, by its SML-Interface and wealth of predefined (documented!) functions, Design/CPN offers the possibility of quickly checking novel ideas. The relatively bad user interface and the complex annotation language make it quite difficult to use at first. For someone familiar with SML (and with some first experience with Design/CPN), however, this is irrelevant.

PEP, on the other hand, contains a number of components that interact mainly via files only (though sometimes via sockets). As the file format is well-documented, it seems reasonably easy to experimentally add new components, or even to reuse parts of PEP in other settings.

6 Summary and outlook

This survey showed an amazing range of high quality tools with very advanced functionality. The contributions of this survey are twofold: firstly, an in-depth survey of high-end tools for hierarchical High-level Petri-nets is given. Doing so, secondly, a detailed catalogue of criteria for the quality of tools has been compiled. This catalogue may form the basis of further surveys. It should be stressed, that this catalogue arose in its final gestalt only in doing this survey, and that well over a third of the overall effort went into setting up this catalogue.

These contributions can be used in different ways:

- as the basis for selecting tools for a specific purpose, as is done in the previous section. Depending on the particular needs of a project, other profiles need to be defined and quite possibly the evaluation-scheme will have to be adapted;
- as a starting point for a more detailed general questionnaire for use at the tools database at DAIMI or for further surveys;
- as an initial statement of requirements for a Petri-net tool useful for anyone interested in implementing such a tool;

The quality of any survey is determined by the amount of data collected, by the scrutiny of evaluation, and by the of the evaluation process. As far as I can see, this is the most thorough and well-documented survey published so far. The usefulness of this report, however, is limited by the restricted number of tools selected in the first round and by the relatively subjective evaluation. Unfortunately, both a broader and a more detailed treatment eliminating possible sources of subjectivity are too much effort to be carried out. Suitable additions and elaborations of the criteria have been added where applicable in the explanations of the tables.

Apart from mere elaboration of the catalogue presented here, special studies on particular aspects of tools (Editor, Simulator, Analysis tools) will have to be conducted as well. Wikarski gives a glimpse of this in his terminological discussion of relevant factors in [27, section 2.1]), and [11] does a quantitative comparison of the analysis components of three tools (INA, PEP, and PROD). For practical applications, the speed of simulation/execution of nets is particularly interesting. Obviously, this field is too large to be discussed or even covered completely in this survey.

Acknowledgements

I would like to thank the participants of the survey for supplying me with evaluation copies, technical support, and feedback on the draft versions of this report. Also, my thanks go to C. Dupuis, and M. Wirsing for legal advice and to C. Maier for proofreading.

References

[1] F. Bause, P. Kemper, and P. Kritzinger. Abstract petri net notation. Forschungsbericht 563, Fachbereich Informatik, Universität Dortmund, 1994.

[2] U. Becker and D. Moldt. Object-oriented concepts for coloured petri nets. In IEEE, editor, *Proceedings: IEEE International Conference on Systems, Man and Cybernetics*, volume 3. IEEE, 1993.

[3] G. Booch. *Object-Oriented Analysis and Design.* Benjamin/Cummings, 1994. Second Edition.

[4] S. Christensen and K.H. Mortensen. Petri nets tools database, 1997. www.daimi.aau.dk/PetriNets.

[5] Commitee ISO/IEC 15909. High-level petri net standard - part 2: Syntaxes, 1997.

[6] Proceedings: 4. Workshop Algorithmen und Petrinetze (AWPN). 1997.

[7] M. Eggert, H. Leszak. *Petri-Netz-Methoden und Werkzeuge. Hilfsmittel zur Entwurfsspezifikation und -validation von Rechensystemen.* Number 197. Springer, 1989.

[8] F. Feldbrugge. Petri net tool overview 1989. In G. Rozenberg, editor, *Proceedings Annual European Workshop: Applications and Theory of Petri Nets '88*, number 424 in LNCS. Springer, 1990.

[9] F. Feldbrugge. Petri net tool overview 1992. In G. Rozenberg, editor, *Proceedings Annual European Workshop: Applications and Theory of Petri Nets '91*, number 674 in LNCS. Springer, 1993.

[10] F. Feldbrugge and K. Jensen. Petri net tool overview 1986. In W. Brauer, W. Reisig, and G. Rozenberg, editors, *Petri Nets: Applications and Relationships to Other Models of Concurrency*, number 255 in LNCS. Springer, 1987.

[11] M. Heiner and P. Deussen. Petri net based design and analysis of reactive systems. In *Workshop on Discrete Event Systems (WoDES'96)*, 1996.

[12] Ivar Jacobson, Magnus Christerson, Patrik Jonsson, and Gunnar Övergaard. *Object–Oriented Software Engineering – A Use Case Driven Approach.* Addison-Wesley, 1992.

[13] K. Jensen and G. Rozenberg. *High-Level Petri Nets. Theory and Application.* Springer, 1991.

[14] E. Kindler. Der Petrinetz-Kern: Ein einfaches Anwendungsbeispiel. In Desel et al. [6].

[15] P. Langner, C. Schneider, and K. Wehler. Prozeßmodellierung mit ereignisgesteuerten Prozeßketten (EPKs) und Petri-Netzen. *Wirtschaftsinformatik*, 39(5):479–489, 1997.

[16] R. Lübeck. Tools Description List, 1995. ls4-www.informatik.uni-dortmund.de/luebeck/pg279/modprak/petri-tools.html.

[17] C. Maier. Objektorientierte Analyse mit Petrinetzen. Diplomarbeit, FB Infomatik, Universität Hamburg, 1997.

[18] C. Maier and L. Mandel. An Introduction to `MOOD`^{2} – Specification and Design of a Live Scoring System for the Compaq Grand Slam Cup 1997. Technical Report 9711, Institut für Informatik, Universität München, 1998.

[19] C. Maier and H. Störrle. Translating Object Charts to High-level Petri-nets. Technical report, Institut für Informatik, Universität München, 1998. to appear.

[20] D. Moldt. *Systemspezifikation mit Petrinetzen.* Dissertation, FB Infomatik, Universität Hamburg, 1996.

[21] Rainerg Neumann. GraphFrame. Ein Framework zur Erstellung von Graphenanwendungen. Diplomarbeit, Uni Karlsruhe, Fakultät für Informatik, 1994.

[22] W. Reisig. *A Primer in Petri Net Design.* Springer, 1992.

[23] H. Störrle. Steps towards a methodology for Software-Engineering for the Internet. Technical report, Ludwig-Maximilians-Universität München, Institut für Informatik, 1997. forthcoming.

[24] H. Störrle. Strukturbisimulation auf höheren Petrinetzen. Diplomarbeit, FB Infomatik, Universität Hamburg, 1997.

[25] M. Trompedeller. Petri net tools, 1993. last update: 1995, www.dsi.unimi.it/Users/Tesi/trompede/petri/alfa.html.

[26] M. Weber. Der Petrinetz-Kern - Eine Aufteilung in Invariantes und Variables. In Desel et al. [6].

[27] D. Wikarski. Petri net tools – A Comparative Study. Technical report, TU Berlin, Fachbereich Informatik, 1997. Bericht 97-4.