Information and Software Technology,
Special Edition on Software Engineering for Parallel Systems; 36(94)7, pp. 435-441.

A Petri Net Based Methodology to Integrate Qualitative and Quantitative Analysis

Monika Heiner
GMD/FIRST
Rudower Chaussee 5
D-12489 Berlin
mh@first.gmd.de

Giorgio Ventre
Univ. degli Studi di Napoli
via Diocleziano 328
I-80125 Napoli
ventre@cps.na.cnr.it

Dietmar Wikarski
FhG/ISST
Kurstraße 33
D-10117 Berlin
dietmar.wikarski@isst.fhg.de

Abstract:
An innovative net-based methodology to integrate qualitative and quantitative analysis of distributed software systems is outlined, and an on-going prototype implementation of a related graphic-oriented tool kit is sketched. The proposed method combines qualitative analysis, monitoring and testing as well as quantitative analysis on the basis of a net-based intermediate representation of the distributed software system under consideration. All transformations (from the distributed software system into a first Petri net model, and between the different kinds of net models) can be done formally, and therefore automated to a high degree. The evaluation of quantitative properties is based on so-called object nets which are obtained by a property-preserving structural compression and quantitative expansion of the qualitative model. Hereby, the frequency and delay attributes necessary to generate quantitative models are provided by the monitoring and testing component.

Keywords:
Parallel software engineering, process-oriented imperative languages, software validation, static analysis, monitoring, testing, performance evaluation, dependability, formal methods, Petri nets, object nets.

1. Introduction

Qualitative as well as quantitative properties of distributed software systems are basically characterized by the interactions between the system’s processes. It is widely accepted that these properties are most naturally analyzed in terms of order and numbers of the events occurring in the system. For this kind of problems, Petri net theory offers a powerful mathematical background.

The approach to software validation based on Petri nets allows to combine the advantages of high-level (specification/programming) languages with those of the Petri net theory. Figure 1 shows the logical architecture of the proposed methodology for net-based software validation. The current physical architecture of a related tool kit is shown in Figure 2. As a collection of different methods or tools, respectively, both figures reflect the belief that a thorough software validation can only be reached by a suitable combination of different techniques complementing each other with respect to the different properties to be validated.

An important property of the Petri net approach is its extreme generality. It aids developers in a general way in reasoning about the behaviour of distributed systems. Because of its generality, it can be applied to distributed systems expressed in a wide variety of specification or programming languages.

The semantics of a particular language are captured by a procedure for automatically deriving Petri net representations for any distributed system expressed in this high-level language. Such a procedure can be implemented as a dedicated compiler, which is in the following shortly called Petri net generator. The modelling of distributed systems by Petri nets resulting in an intermediate representation yields several advantages.

First, tools for analysis and reduction extract information about events and their ordering directly from this intermediate representation of the system. Therefore, they do not rely on any special assumptions, but regard general features of distributed systems. They can be applied to any system once a Petri net representation for the system has been obtained.

Second, due to the generality of the Petri net approach, tools based on Petri nets can be extended to provide common analysis methods across a number of phases in the software development process as soon as some formalized description of the distributed system under development is available. This might be a source of valuable commonality and integration in a software development environment.

Third, the net-based intermediate program representation serves as a common root from which different net-based software validation methods, basically on the communication/synchronization level, are able to start, e.g.

- analysis of qualitative properties like context checking of static semantics and verification of functional behaviour (see paragraph 2.),
- monitoring and testing (see paragraph 3.),
- analysis of quantitative properties (see paragraph 4.).

These different validation methods require net models which vary partly in their level of abstraction (e.g. granularity of considered control and/or data flow, delay and branching information). However, the transformations starting from the common net-based intermediate program representation can be done to a high degree formally and therefore automated. This includes especially the innovative approach of an as far as possible formal transformation of the qualitative model into a quantitative one.

2. Net-based Qualitative Analysis

Net-based qualitative analysis is a well-known approach since more than ten years /Heiner 80/. But to put it into practice is still a promising challenge, because of the analysis algorithms' complexity and the fact that the results gained in the analysis phase are very sensitive to the abstractions done in the modelling phase. A related tool kit consists basically of the following four logical components: Petri net generator, linker, analyzer and error reporting and interpretation system /Heiner 92/.

The Petri Net Generators in use (PNG_{PDL} /Koenig 85/, /Grzegorek 91/ and PNG_{C} /Czichy 92/) produce for a given sequential process Petri net representations of its (reduced/non-reduced) control structure. Besides this basic functionality, further information is supplied, which allows

- an automatic layout of the generated net afterwards,
- an assessment of the program's structural complexity (Number of Acyclic Paths /Heiner 88/).

After that, the graphical Petri net EDitor PED /Czichy 93/ is used for visualization and linking of the generated Petri net parts, and for adding any possibly required supplements, e.g.

- modelling of all control variables and related operations, improving the generated control structure model to a control flow one (the automatization of this step is in preparation),
- modelling of the system environment behaviour,

- fault models (e.g. message loss or duplication induced by erroneous message channels).

Finally, the filtering capabilities of PED are used to output the particular data structures required by the analysis tools.

All used analysis tools (INA /Starke 92/, PROD /PROD 92/) have to be understood as implementations of well-proved theorems of Petri net theory. As a consequence, the results obtained by these tools can only be in terms of Petri net theory independently of any underlying special semantics of the net or software under test, respectively.

As an advantage, Petri net theory offers different methods to analyze qualitative aspects of distributed systems, e.g.

- prototyping by playing the token game,
- static analyses by net reduction, structural analysis or net invariant analysis,
- dynamic analyses by complete/reduced construction of the system's state space (reachability graph), possibly followed by
- model checking to evaluate the temporal relationship of logic formulae.

Net-based **prototyping** aims at simulation of the functional behaviour by playing the token game. The results gained depend on the abstraction level of the underlying net model. But in any case, prototyping is only a confidence-building approach unable to replace exhaustive analysis methods.

All **static analysis** techniques have in common that they avoid the construction of the reachability graph. While reduction and structural analysis aim at context checking (of general semantic properties, which have to be fulfilled by any program independent of its special functionality), the invariant analysis corresponds mainly to verification (of special semantic properties, which are determined by the intended special functionality). The net based proof of program invariants by showing the existence of related net invariants shows some similarities to the classical approach of axiomatic program verification. First, suitable program invariants have to be hypothesized, and second, the related net invariants have to be found from the (in general non-minimal) basis of invariants provided by a net analysis tool.

Dynamic analysis techniques have to be used if the static analysis efforts were not successful or if properties are wanted, which cannot be analyzed statically at all (e.g. freedom of dynamic conflicts, firing of facts etc.).

In **model checking**, the reachability graph of a Petri net is interpreted as data base, and temporal logic is used as query language for asking questions over it. Therefore, all properties which can be expressed in the used version of temporal logic (CTL - Computation Tree Logic /Clarke 86/) can be checked. By this way, also very large reachability graphs become manageable.

The error reporting and interpretation phase requires at first the retranslation of the analysis results in terms of Petri nets into terms of the software under investigation, which can be done automatically only if all net reductions are carried out very carefully. Decisions about the truth and gravity of any analyzed failure situations remain the intellectually demanding duty of the software quality assurance team.

The methods mentioned imply partly the use of higher-level net classes (at least as short-hand notation of lengthy place/transition nets). But we still try to restrict ourselves to such Petri net classes which support some type of exhaustive analysis (as opposed to simulation of the functional behaviour).

3. Net-based Monitoring and Testing

Static analysis techniques for distributed software show the property of allowing an „a priori“ evaluation of the inherent characteristics of an algorithm. The main advantage of adopting such kind of techniques is that, depending on the properties of the technique itself, information at a high level of abstraction can be produced about the expected behaviour of the application. This abstraction capability, however, is also the origin of the main disadvantage of these approaches. A number of authors have shown in the years how difficult it is for static analysis techniques to extract parameters related to the behaviour at run-time of a program when such behaviour is not completely expressed by the source code (see e.g. /Kumar 88/, /Ghosal 91/).

For example, the adoption of a Petri net based technique for program analysis can find limitations in its application whenever data-dependencies not reflected in the model characterize the algorithm under examination.

On the other hand, „a posteriori“ techniques are available for program testing, i.e. based on the analysis of data collected during real, monitored executions of a program. These techniques, though capable of extracting with the highest precision information related to the actual behaviour of an algorithm, are characterized by other limitations. First, in order to have information about the program behaviour which is general enough, a number of different tests have to be conducted, in order to produce a sufficient number of case studies. This is particularly true in the case of data-dependencies or nondeterminism. Second, the huge amount of data produced during the monitored sessions can be difficult to analyze to extract general informations about the algorithm itself.

On the basis of these considerations we decided to integrate these two techniques in order to overcome the difficulties that characterize them when used separately. The underlying idea is based on the fact that a program model produced by a static analysis technique can be notably improved if „tuned“ by using real data extracted during monitored executions.

We believe that the availability of information related to the actual behaviour (or to some of the possible behaviours) of a parallel program can be used to resolve some of the ambiguities that can be generated in a model produced by taking into account only the source code of a program.

As an example, we consider the use of a Petri net based methodology to determine the inherent parallelism for a distributed application. The solution of such a problem might be helpful in evaluating the right number of processors to assign to a certain application that has to be executed on a multicomputer.

The approach proposed in /Ghosal 91/ for the solution of this problem is based on the determination of the processor working set, i.e. the maximum number of processors that can be actually used in parallel by a distributed algorithm. This parameter is dependent not only on the algorithm structure but, in several cases, also on the dimension of the input data and on the presence of nondeterministic behaviour. However, the technique proposed in /Ghosal 91/ is based only on the measurement of the speed-up obtained by a program by varying the number of processors assigned to it. In order to solve the problems related to dependencies on the input data size, a large number of monitored execution has to be performed by varying the input data dimension and values.

We believe that also for this example, the adoption of a mixed technique for program analysis can be extremely helpful. In fact, by simply analyzing the program structure with a Petri net based approach number of parameters related to its inherent parallelism can be extracted. This information can then be tuned and completed by integrating it with the data produced by a limited number of monitored executions, to solve the ambiguities connected to nondeterminism and input data dependencies.It is our opinion that such procedure of **net-based monitoring** can effectively provide quantitative parameters like delay and frequency attributes which have to be added to the qualitative model in order to form what we call a quantitative net model. Such information can be selectively collected by positioning appropriate probes in the software to be monitored, using the net model itself to individuate the most appropriate points to control. In addition to reduce the amount of information to collect (and analyze), this selectivity also limits the additional load that the monitoring software can induce on a system.

A similar effect can of course be attained by using proper functional simulation techniques and tools, capable of simulating efficiently all the different components of a distributed system. For this reason, the functional simulator (like Galileo /Knightly 92/) can be even integrated as an additional tool provided by the analysis tool kit.

The integration of a Petri Net based tool and monitoring software can be used also in the testing phase of a program. The availability of a high-level model of a program based on Petri nets can be exploited to simplify the testing particularly for what concerns communication and interaction among processes. We define **net-based testing** as the activity of integrating qualitative modelling and quantitative information towards systematic testing of distributed software.

The idea of integrating qualitative analysis and (high-level) debugging has been originally proposed in /Dahmen 89/ to limit the time-dependent nondeterminism in distributed software, well-known in the debugging community as probe effect /Gait 86/. The basic idea of such approach is the integration of net-controlled analysis with a debugging tool based on the **instant replay** mechanism. Instant Replay /LeBlanc 87/ is a general approach to high-level debugging making the behaviour of distributed programs reproducible. A further aim is to reduce the influence of monitoring tools on the expected behaviour of a program. This last requirement is particularly important in the case of programs characterized by nondeterminism.

The proposal, originally proposed for controlling shared memory data structures, has been extended to distributed memory architectures (e.g. /Dahmen 89/, /Mazzeo 92/, /Wheil 91/). When a distributed program is executed the relative order of inter-process communication events is recorded (**record mode**), for each process, in a so-called process history tape (shortly process trace). To reduce the probe effect, only information to identify a communication event is recorded, and not the event itself. The program can then be re-executed (**replay mode**), by using the process history tapes to enforce the execution order of the communication events to be the same as previously recorded. In this way the behaviour of the program has been made reproducible without the need of recording information related to a global time.

The approach we propose is a net-based methodology for a systematic test of a distributed program combining the information produced by a non-intrusive monitor tool with the information we do have due to the Petri net model generated for the purpose of quantitative analysis. The combination works in two opposite directions.

- The instant replay technique monitors and records the process execution traces. One method to evaluate these traces (process history tapes) is to visualize in the corresponding (sub-) reachability graph of the process system's Petri net model, which system state has been reached and via which execution path the system has been gone through.

  The reproducibility of the program execution and its independence from time ensures that this kind of analysis can be interrupted or suspended without endangering the correctness of the procedure.

- On the other hand, having the Petri net model, we are able to synthesize a new "process history tape" and to use it to induce a specific behaviour in a distributed program,

  i.e. to enforce the process system

  - to follow a certain execution path
    supporting a systematic and maybe complete test expressible by an appropriate complexity measure, or testing whether anomalies reported by qualitative analysis are actually faults or non-faults, and

- to reach a certain system state
  where, for example, another debugging session might be started from. This allows to verify the behaviour of a system in a particular system state by forcing the system to reach it and then by letting him evolve freely.

Finally, Petri net theory can be used to explain efficiently the basic principles the instant-replay method is based on (for related details see /Dahmen 89/).

4. Net-based Quantitative Analysis

A recently widely used class of formal models for quantitative analysis (i.e performance and reliability prediction based on a formal model) of distributed software systems are timed and stochastic Petri nets. Hereby, one of the most popular and powerful classes is that of Generalized Stochastic Petri Nets (GSPN, see e.g. /Ajmone 84/). A useful straightforward generalization to allow also deterministic delays of transitions are Deterministic and Stochastic Petri Nets (DSPN, see e.g. /Ajmone 87/). The corresponding modelling and evaluation tools GreatSPN /Chiola 91/ and DSPNexpress /Lindemann 94/ are intended to be used (as one possibility) for quantitative analysis in the given tool set. Hereby, the main advantage of the new tool DSPNexpress compared with the proved GreatSPN (which allows a timed interactive simulation of DSPN) is the availability of an efficient numerical algorithm for computing steady-state solutions of DSPN.

Unfortunately, GSPN, DSPN and the other known classes of timed and stochastic Petri nets assume information about a global state and thus, also a global time axis. These assumptions are closely connected with the (global) „must“ firing rule. That is, any transition, if enabled, is forced to fire: either immediately or after certain prescribed -- possibly stochastic -- amount of time. In the case of conflicts between transitions, they are solved either according to (global) priorities or according to given (global) weights interpreted as probabilities. A consequence of such a net semantics (in particular, of the global „must“ firing rule) is that qualitative properties of an untimed net model (boundedness, liveness) are in some cases not preserved in a timed or stochastic net model with the same structure.

As an example, consider the net depicted in Figure 3. In the untimed case, this net is live, but has the unbounded place s. After changing the firing rule to the „must“- one, the transition t becomes a dead one, but the place s will be bounded. The change of the firing rule in this model may be also considered as a change to a (discrete) timed net, whereby all transitions are interpreted as a timed transition with the delay of one.

To avoid this effect of changing qualitative properties by introducing time into nets, an alternative, new class of net models is intended to be used in our approach. The key idea of this class, called **locally Markovian Object Nets (MON)** /Wikarski 92/, is to decompose a Petri net into such subnets (so-called objects), which exhibit among others the following property: Any two transitions being in a static conflict, together with all of its preplaces, belong to the same object.

The behaviour of a MON is determined by the following transition rules.

1. Conflicts between enabled object-local[1] steps of transitions are solved according to given probabilities.
2. „Internal“ transitions (such which are not directly connected with places of other objects) are governed by the so-called object-local „must“-firing rule, and „external“ transitions (such transitions which are connected to places of other objects) follow the usual „may“ firing rule.

1. Hereby, „object-local“ means „global with respect to an object“- in contrast to „global with respect to the whole net“ as above.

Objects may be considered as a new type of locality in a net, which is greater than the traditional one (a transition), but usually less than the whole net. The real-world entities corresponding to the places and transitions of the same object are assumed to be observable in common. Therefore, steps of transitions may be equipped with object-local probabilities (or relative frequencies, resp.) of conflict solution between steps and with object-local “time“, which is based on the object-local „must“-firing rule. More precisely, given a marking of an object, a maximal set of concurrently enabled internal transitions (i.e. a maximal step) must be fired immediately when this marking has been reached. Hereby, conflicts between steps have to be solved in accordance to given probabilities or weights of the steps. On the other hand, all external transitions follow the usual firing rule, i.e. if enabled, they may fire, but need not to do so. As a consequence, the qualitative net properties once obtained for the global net behaviour (which is determined by the external transitions) will be preserved also in the case of its quantitative evaluation after this introduction of the object-local must firing rule into objects of a net.

To make this idea more transparent, consider once more the net of Figure 3. An admissable decomposition into objects is given by declaring the transitions t, t1, t2, t3, t5, t7, t8 as external ones and the complement as internal ones. The corresponding objects are characterized by the sets of transitions (t, t1, t2), (t3), (t4,...,t7) and (t8,...,t12). As it can be seen easily, by changing the firing rule for all the internal transitions t4, t6 and t9,...,t12 to the „must“ one, the liveness and boundedness properties of the net do not change.

Due to the (to some extent) free choice to decompose a given net into objects, local time scales can be chosen in accordance to the possibilities of common observation of events. Moreover, due to the possibility to see „time“ as quantitative abstractions of the behaviour of objects, the size of each object may be chosen according to the required or necessary time abstraction of the model.

The information obtained by observing or testing the local behaviour of objects (in particular: the relative frequencies of conflict decisions in favour to certain sets of concurrently enabled transitions, given the state of the object) is used to simulate and thus to predict the global system behaviour on the basis of local information. In view of the inherently distributed character of the model (which is able to reflect adequately the distributed character of the modelled system), the most appropriate use of object nets for quantitative evaluation of distributed systems is as a conceptual model for a distributed MON simulator instead of the use of classical methods of (global) analytical solvers or simulators (see Figure 2).

5. Final Remarks

Validation methods can be classified according to the properties they aim at. The different methods do not compete, but complement each other. A recommended order of validation methods takes into account that

- validation should be applied as early as possible,
- proper functionality is a prerequisite for an evaluation of quantitative properties,
- the expected functionality can only be guaranteed in any case if all consistency conditions of context checking have been fulfilled.

We believe that some of the validation methods and tools, which are currently only separately available, can be successfully combined to provide an integrated software development environment. Petri nets provide an adequate common basis for such a tool kit that extensively supports different methods of dependable distributed software engineering, because Petri nets are a suitable intermediate representation for

- different phases of software development cycle,

- different languages, and
- different validation methods.

Our investigations are greatly influenced by the goal of providing a workbench which can be applied by an engineer engaged more in software quality assurance than in Petri net theory. But, a thorough software validation is expensive and requires an adequate mathematical foundation.

In our opinion, the level of practicability of a method like this is strongly influenced by the level of an engineer-like preparation of the tool kit provided. To fill the gap between theory and application, a prototype implementation of a graphic-oriented tool kit supporting the proposed validation methodology is under development. First results are available for SUN workstations with X11R4/Motif Interface /Czichy 93/.

6. References

/Ajmone 84/:
Ajmone Marsan, M.; Balbo, G.; Conte, G.:
A Class of Generalized Stochastic Petri Nets for the Performance Analysis of Multiprocessor Systems;
ACM Trans. Computer Syst. 2(84)1, pp. 93-122.

/Ajmone 87/:
Ajmone Marsan, M.; Chiola, G.:
On Petri Nets with Deterministic and Exponentially Distributed Firing Times;
in: G. Rozenberg (Ed.) Advances in Petri Nets 1986, LNCS 266, Springer Berlin 1987, pp. 132-145.

/Chiola 91/
Chiola, G.:
GreatSPN 1.5 Software Architecture;
in Proc. 5th Int. Conf. on Modelling Techniques and Tools for Computer Performance Evaluation, Torino, Italy, 2/1991, pp. 117-132.

/Clarke 86/
Clarke, E. M.; Emerson, E. A.; Sistla, A. P.:
Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications;
ACM Trans. on Programming Languages and Systems 8(86)2, pp. 244-263.

/Czichy 92/
Czichy, G.:
Implementation of a Petri Net Generator for INMOS C Programs (in German);
Techn. Report of Practical Studies, GMD/FIRST, Berlin 2/1992 .

/Czichy 93/
Czichy, G.:
Design and Implementation of a Graphical Editor for Hierarchical Petri Net Models (in German);
Diploma Thesis, TU Dresden und GMD/FIRST, 6/1993.

/Dahmen 89/
Dahmen, J.; Heiner, M.:
An approach to Systematic Testing of Distributed Software (in German);
Proc. Problemseminar “Programmiersysteme für Mikrorechner”, Bad Saarow, 11/1989, Informatik-Informationen-Reporte 5(1989) 11, pp. 67-81.

/Gait 86/
Gait, J.:
A Probe Effect in Concurrent Programs;
Software Practice and Experience 16(86)3, pp. 225-233.

/Ghosal 91/
Ghosal, D.; Serazzi, G.; Tripathi, S.K.:
The Processor Working Set and Its Use in Scheduling Multiprocessor Systems;
IEEE Transaction on Software Engineering 17(91)May, pp. 443-453.

/Grzegorek 91/
Grzegorek, M.:
Further Development of a PDL/D Compiler (in German);
Techn. Report of Practical Studies, IIR/AdW, Berlin 1/1991.

/Heiner 80/
Heiner, M.:
A Contribution to Deadlock Analysis Based on a Language-guided Programming Methodology (in German);
Ph. D. Thesis, TU Dresden, 9/1980.

/Heiner 88/
Heiner, M.:
A Complexity Measure of Distributed Programs;
Proc. 2nd Int. Seminar on Modelling and Performance Evaluation, Wendisch-Rietz, 11/1988, Informatik-Reporte/IIR 17/88, pp. 72-83.

/Heiner 92/
Heiner, M.:
Petri Net Based Software Validation, Prospects and Limitations;
ICSI-TR-92-022, Berkeley/CA, 3/1992.

/Knightly 92/
Knightly, E.W.; Ventre, G.:
Galileo: a Tool for Simulation and Analysis of Real-time Networks;
ICSI-TR-93-008, Berkeley/CA, 3/1993.

/Koenig 85/
Koenig, H.; Heiner, M.; Onisseit, J.:
Defining Report of the Protocol Description Language PDL (in German);
Techn. Univ. of Dresden, Dep. of Informatiques, Research Report TU 08 RS-L/LN-K5/015, 1985.

/Kumar 88/
Kumar, M.:
Measuring Parallelism in Computation-Intensive Scientific/Engineering Applications;
IEEE Trans. on Computers 37(88)Sept., pp. 1088-1098.

/Leblanc 87/
Leblanc, T.; Mellor-Crummey, J. M.:
Debugging Parallel Programs with Instant Replay;
IEEE Trans. on Computers 36(87)4, pp. 471-482.

/Lindemann 94/
Lindemann, C.:
DSPNexpress: A Software Package for the Efficient Solution of Deterministic and Stochastic Petri Nets;
to appear in Performance Evaluation, 1994.

/Mazzeo 92/
Mazzeo, A.; Savy, C.; Ventre, G.:
A High Level Monitor for Parallel Systems;
in Messina, P.; Murli, A. (eds): Parallel Computing: Problems, Methods, and Applications; Elsevier Science Publishers, 1992, pp. 531-542.

/PROD 92/
PROD - User Manual;
Helsinki Univ. of Technology, Digital Systems Laboratory, 1992.

/Starke 92/
Starke, P. H.:
INA - Integrated Net Analyzer, Manual (in German);
Berlin 1992.

/Wheil 91/
Wheil, W. et al.:
PRELUDE: A System for Portable Parallel Software;
Technical Report, MIT/LCS/TR-519, Massachusetts Institute of Technology, Laboratory for Computer Science, October 1991.

/Wikarski 90/
Wikarski, D.:
Object Nets - A Canonical Class of Models for Behaviour Simulation and Structure Synthesis of Distributed Systems?
in Proc. 3rd Int. Seminar on Modelling, Evaluation and Optimization of Dependable Computer Systems, Wendisch Rietz, 11/1990, Informatik-Informationen-Reporte 6(90)12, pp. 91-100.

/Wikarski 92/
Wikarski, D.:
Locally Markovian Object Nets - The Conceptual Model;
Proc. Int. Seminar on Concurrency, Programming and Specification (CSP 92), Humboldt-University Berlin, Berlin 11/1992, pp. 182-189.

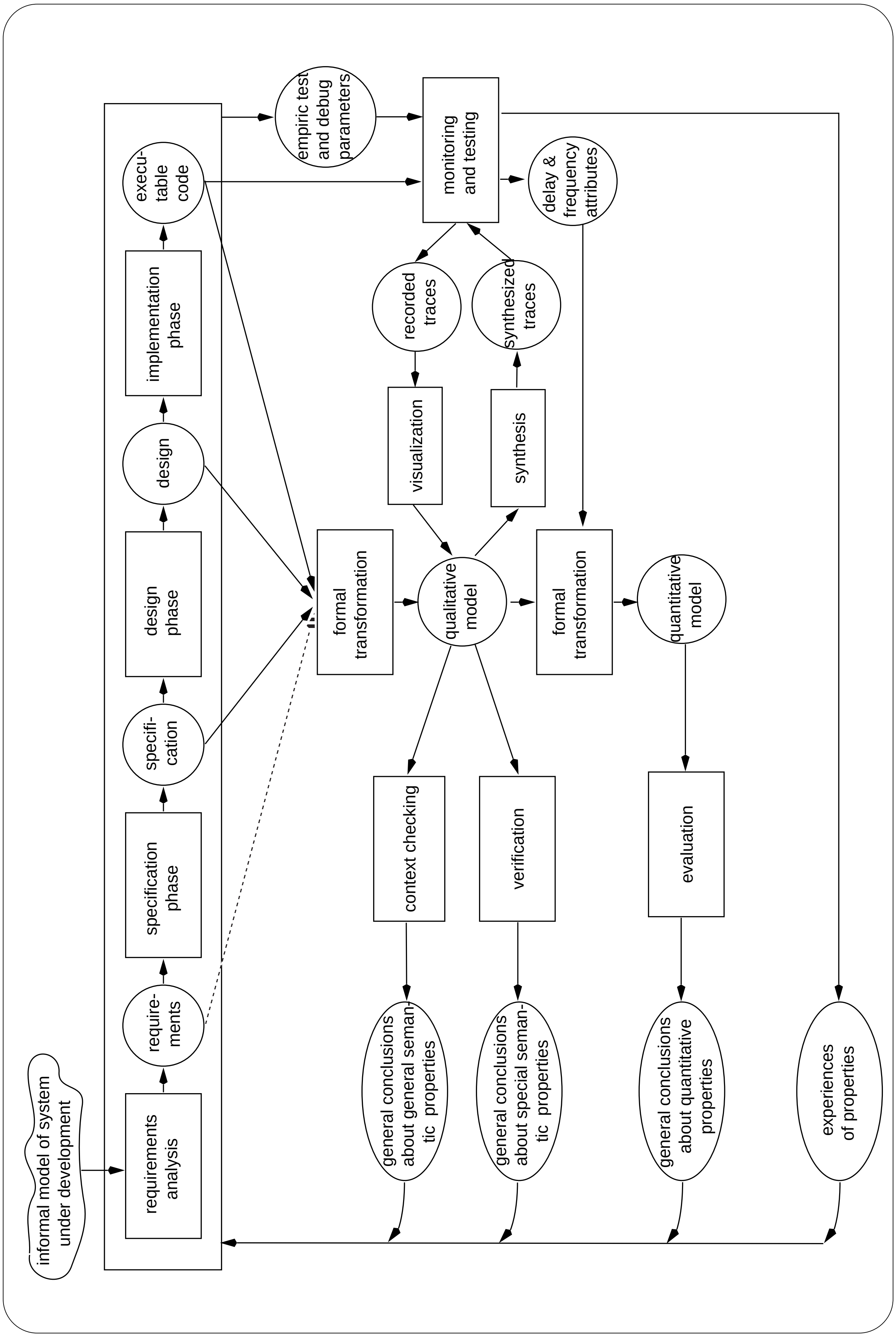


Figure 1: Overview of the net-based methodology.

Figure 2: Physical Architecture of Net Based Tool Kit:

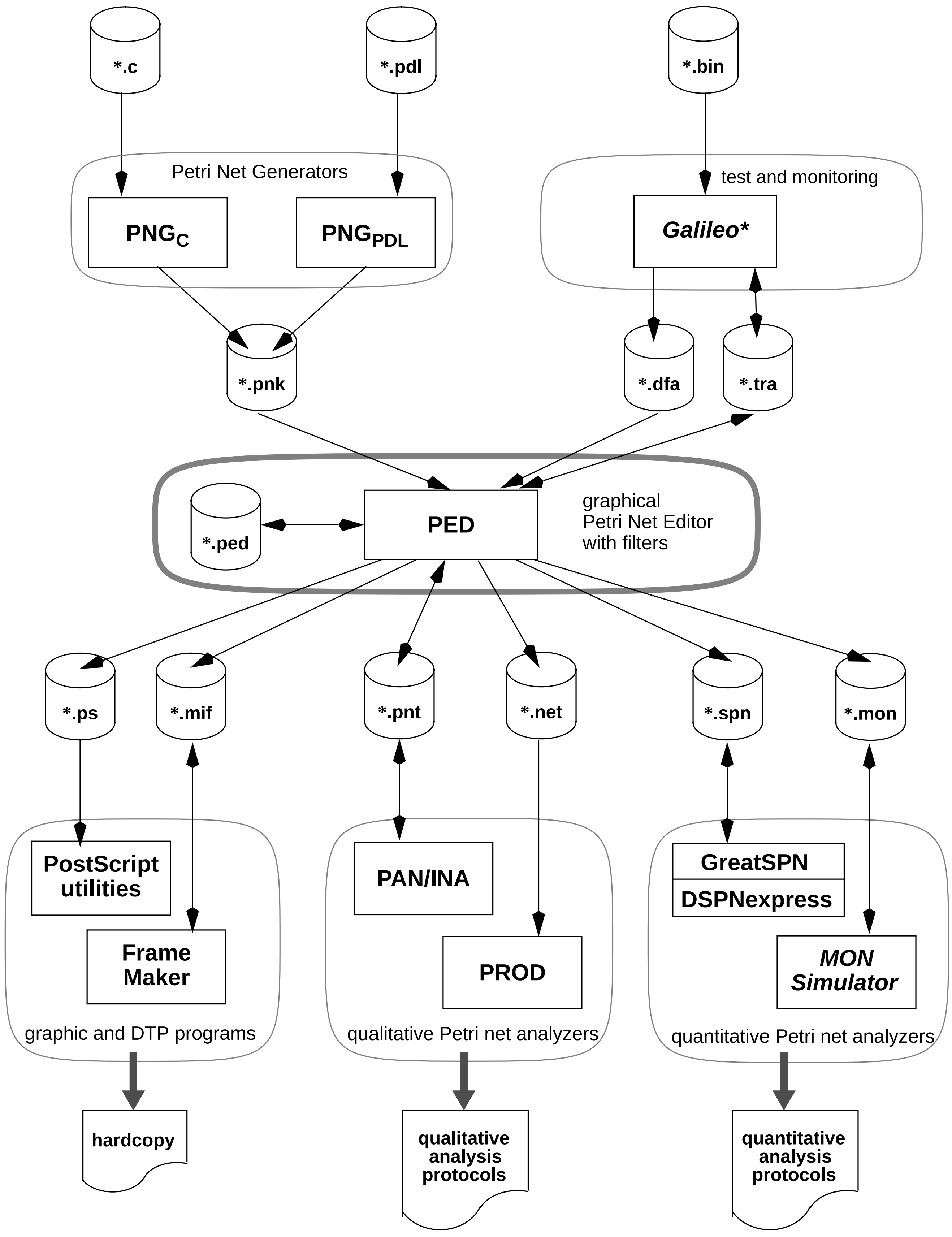
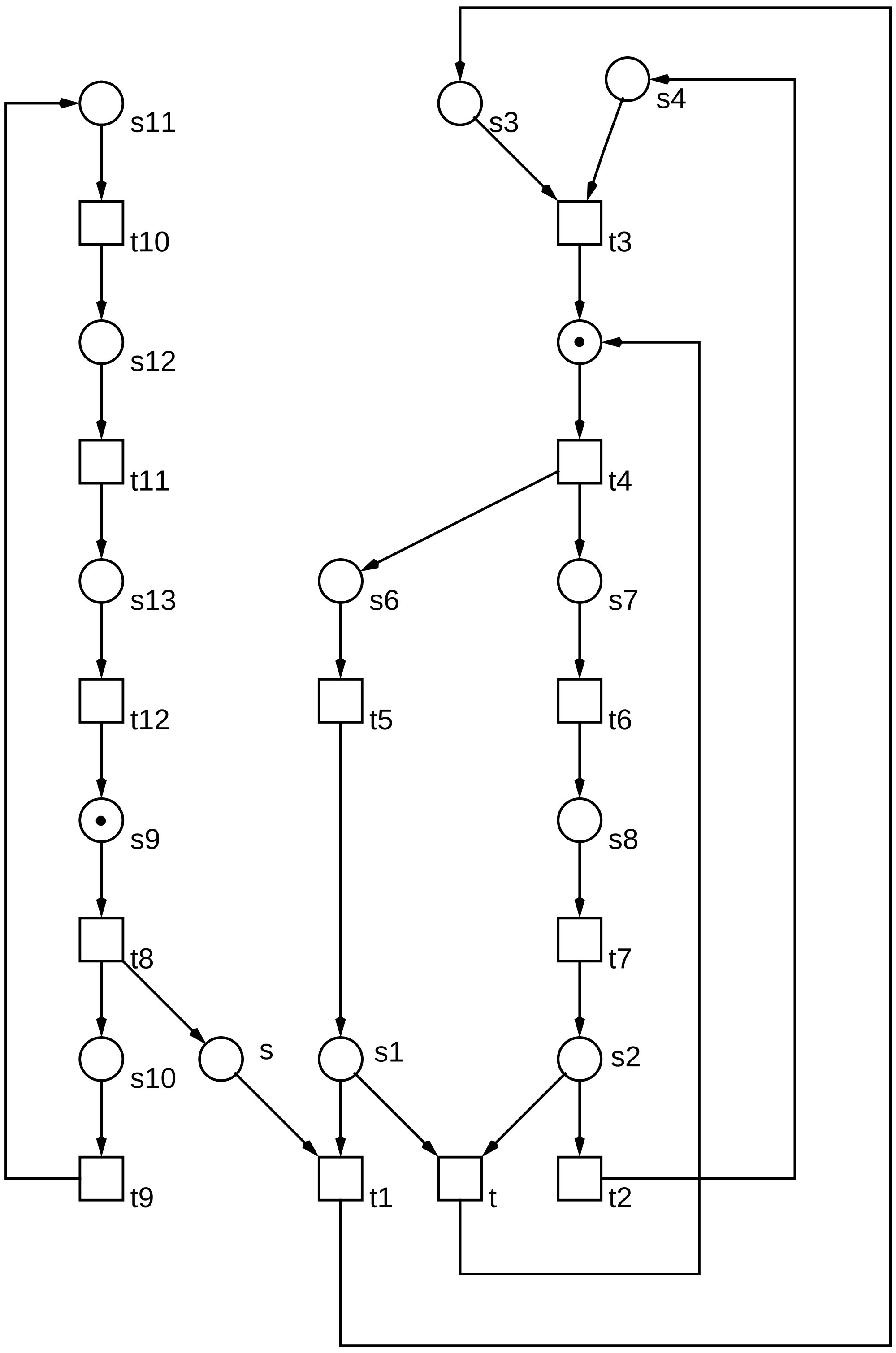


Figure 3: An example net.


s11
s3
s4
t10
t3
s12
t11
t4
s13
s6
s7
t12
t5
t6
s9
s8
t8
t7
s10
s
s1
s2
t9
t1
t
t2