

Stochastic Analysis of CSP Specifications Using a CSP-to-Petri Net Translation Tool: CSPN

Frederick T. Sheldon¹

The University of Texas at Arlington

Abstract

An experimental tool and approach has been developed to explore the *specification and analysis of stochastic properties for concurrent systems expressed using CSP* (communicating sequential processes). The approach is to translate a formal system description into the information needed to predict its behavior as a function of observable parameters. The idea uses a theory based on proven translations between CSP and Petri nets (PNs). In particular, the tool translates the design specification, written in a textual based CSP dialect named P-CSP, into stochastic Petri nets for analysis based on the structural and stochastic properties of the specification. The grammar and CSP-to-Petri net (CSPN) tool enable service and failure rate annotations to be related back into the original CSP specification. The annotations are then incorporated in the next round of translations and stochastic analysis. The tool therefore automates the *analysis and iterative refinement of the design and specification* process. Within this setting, the designer can investigate whether functional and non-functional requirements can be satisfied.

Keywords: Specification modeling, dependable systems, process algebras, Petri net and software tool.

Table of Contents

1.	Introduction	3.3.	Mapping CSP to Petri nets using canonical translation rules
1.2.	Predicting the performability of formal specifications	4.	Some CSPN mechanics
1.3.	Survey of related work	4.1.	Co-matrix expansion
1.4.	Methodology	4.2.	CSP represented as a network structure
2.	CSPN tool overview	5.	Summary
2.1.	Translation phases of the CSPN tool	5.1	Evaluation
2.2.	Running the CSPN tool	5.2	CSPN specifications
2.3.	CSPN data structures	6.	References
3.	The CSP-based language (P-CSP) primitives	7.	Appendix
3.1.	Stochastic Petri nets		
3.2.	SPNP and the C-based Stochastic Petri net Language (CSPL)		

1

¹All correspondence should be addressed to Frederick T. Sheldon, Dept. of Computer Science and Engineering, The University of Texas at Arlington, Arlington, TX 76019-0015, email: sheldon@cse.uta.edu. Sheldon has been supported since 1993 by a NASA Graduate Fellowship from Langley Research Center (#NGT-50896).

1. Introduction

Today's computing systems are large and complex. Therefore, informal and intuitive specifications are too vague and imprecise to capture the complete semantics of a system's requirements. A formal specification language is founded on mathematical principals and is used to describe system properties and to provide a systematic approach to avoid ambiguity, incompleteness and inconsistency [16, 19, 33]. Current systems must also have high performance and reliability. Formal specifications provide good support for designing a functionally correct system. They are weak at incorporating non-functional performance requirements (like reliability). Techniques which utilize stochastic Petri nets (SPNs) are good for evaluating the performance and reliability of a system. However, they may be too abstract or cumbersome from the stand point of specifying and evaluating functional behavior. Therefore, one major objective for developing the CSPN tool is to provide an integrated approach to assist the user in specifying both functionality (qualitative: communications, controls, redundancies) and performance (quantitative: reliability and execution deadlines). In this way, the merits of a powerful modeling technique for performability analysis (using SPNs) can be combined with a well-defined formal specification language. In doing so, we can come closer to providing a formal approach to designing a functionally correct system that meets reliability and performance goals.

1.2. Predicting the performability of formal specifications

This approach is based on the notion that formal, mathematically precise methods should be used to design complex, safety critical systems [5, 31]. Thus, given a formalized functional specification of a system and its external constraints (e.g., failure rates, communication delays, synchronization dependencies, deadlines), what mechanisms are available for avoiding or tolerating faults/errors and how do they impact the performance and reliability (i.e., performability) of the system [23-25, 35, 36]?² As specifications are refined into detailed designs, the reliability and performance requirements can also be refined to reveal the trade-offs in design alternatives such as deciding –what are the critical system elements; –what features of the system should be changed to improve the system's reliability; –verifying and/or validating performance and reliability goals using stochastic system models.³ Thus, in this approach, the critical components of the requirements specification are abstracted and a system is specified using the P-CSP language.⁴ Once the specification has been translated, we estimate model parameters. At this point it is easy to introduce timing constraints among feasible markings of

²Some examples showing the approach are available in [36, 37].

³The term validating used here is in the sense of determining whether the design specification satisfies the intent of performance and reliability requirements. Verify is not used because it connotes a correctness determination, and is better used when matching requirements (or design specifications) to implementation.

⁴The CSP-based grammar does not restrict us from considering correctness properties; however, we are interested only that the structural properties be preserved.

the net and to employ SPNP for stochastic Petri net analysis [8-14].⁵

1.3. Survey of related work

Wang presents a procedure (which could be automated) for transforming an Estelle specification into a Stochastic Reward Net (SRN) formalism [42].⁶ The objective of transforming Estelle into an SRN is to have a system designer specify a system using Estelle and then the specification is automatically transformed into an SRN to carry out the performance and reliability analysis [34, 41]. Donatelli et al., have developed EPOCA which is an integrated Environment for analysis and Performance evaluation Of Concurrent Applications. The analysis is based on stochastic Petri nets (GSPN) starting from a concurrent program written in DISC (DIStributed C), an extension of C to include concurrent constructs of the CSP type. In EPOCA a GSPN model is automatically generated from the DISC program [17-18, 3-4]. The EPOCA environment including a rich graphical user interface was finalized and delivered in 1995.

Others including Davies and Schneider describe the language of real-time CSP used to specify reactive systems in terms of their communicating behavior. Each system component is represented as a process that shows where communication takes place. By combining processes, a description of the system in terms of its components is produced [15]. Peleska gives a formal method based on CSP to design fault tolerant systems combining algebraic and assertional techniques to formally verify correctness properties. Liu and Joseph give a method for transformation of programs constructed for a fault-free system into fault-tolerant programs suitable for execution on a system susceptible to failures [27]. Priami gives a technique for integrating behavioral and performance analysis with topology information using Stochastic pi-calculus [32]. VanGlabbeek gives a structural operational semantics of PCCS as a set of inference rules which constitute a semantic mapping from the set of process expressions to a particular domain of probabilistic labeled transition systems [40]. Current trends have evolved in theory and practice to enable improved methods for ascertaining a specification's dependability and performance, their interactions and corresponding tradeoffs [1-2, 6-7, 12-14, 32-34].

1.4. Methodology

This work offers an approach to predicting system behavior (in terms of reliability and performance) based primarily on the structural characteristics of a formal functional specification. The CSPN tool and methodology is based on a sound formalization of CSP which provides process constructors, including primitives for parallel and sequential composition, nondeterministic choice, and recursion. To support top-down development, the grammar and CSPN tool provide a notion of refinement that allows a designer to describe a system at an

⁵SPNP is "Stochastic Petri Net Package" and was developed at Duke University (see [8, 11]).

⁶Estelle is an ISO standard formal specification language and SRN is a well-developed modeling technique that is used to carry out performance and reliability analysis.

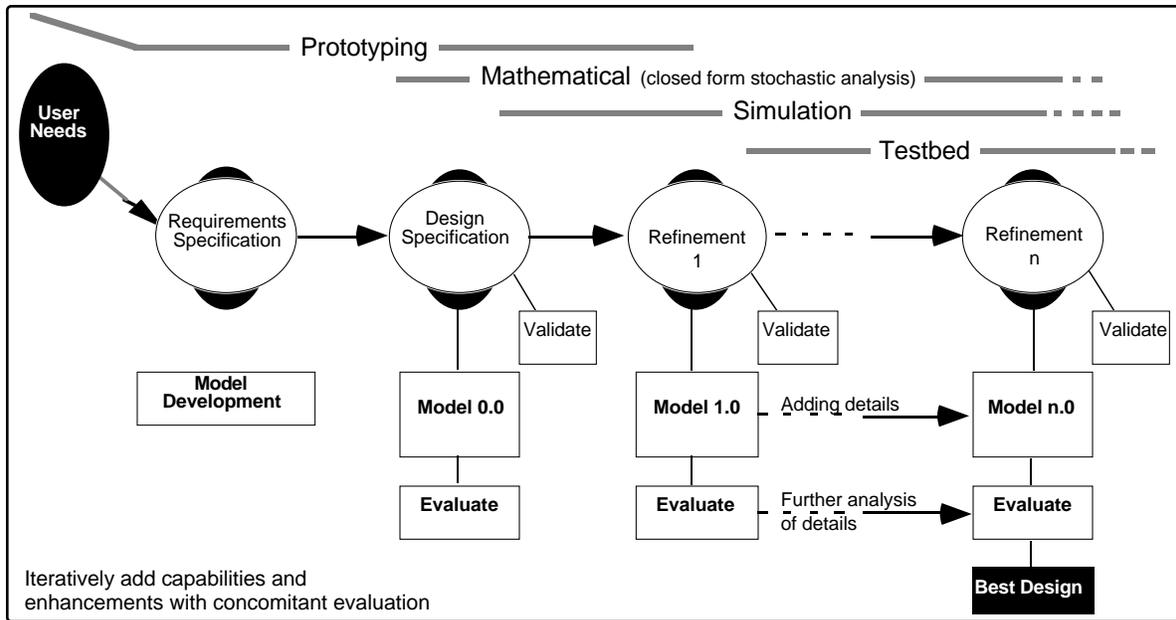


Figure 1. Refinement of system models.

appropriate abstract level. At this level, a designer may estimate the values of non-functional requirements (so called budgeting). Later, the designer may add more details by showing the internal structure of a component, explicitly presenting local communications, and modifying the budget. It is important to facilitate systematic refinements and then re-evaluate their impacts to optimize design parameters. Figure 1 shows how this approach advocated involves abstraction from the requirements specification into a design specification and then further evaluations based on stochastic analysis of the system models.⁷ The CSPN tool is used in the context of the 7 steps listed in Table 1.

2. CSPN tool overview

The core augmentation to existing approaches is provided by the CSP-based grammar and canonical CSP-to-Petri net translation rules for process composition and decomposition. The translation rules are codified in the CSPN tool. In brief, the mechanism consists of decomposing individual CSP constructions into canonical Petri net structures. The elemental structures are linked together in a hierarchical fashion according to their adjacency and nesting within the CSP specification. Once CSPN has created this network of linked structures it traverses the net and expands the process descriptions which are represented as sub-Petri nets into larger and larger nets. Also, as CSPN decomposes the CSP constructions, it identifies and records service and failure rate annotations which are later incorporated in the SPNP specification file. When CSPN encounters failure annotations (and the "-f" command line option is set), it creates supplemental

⁷Automatic translation of the design specification into a stochastic Petri net representation enables the use of a good number of sophisticated design and analysis tools which are based on Petri nets.

TABLE 1 METHODOLOGY: STEPS FOR SPECIFICATION AND ANALYSIS

Step	Description of steps used in the approach.
1.	Abstract the critical elements of the requirement specification and formulate a CSP specification for the system under study.
2.	Translate between CSP and Stochastic Petri nets.
3.	Assign performance and reliability parameters among subsystem components.
4.	Analyze the Petri nets for stochastic properties [using SPNP] (validate performance and reliability goals using stochastic system models).
5.	Decide what features of the system should be changed to improve the system's reliability (and/or other stochastic properties, e.g., performance).
6.	Augmentation: relate stochastic properties back to top level (CSP) specifications (e.g., failure rates, service rates, error handling).
7.	Understand the effect these non-functional requirements have on <i>cost</i> .

failure transitions. After the preliminary structure of the Petri net is complete, CSPN must reconcile synchronization points because CSP input/output actions must rendezvous at a particular point which is translated into a transition that is named by the message that is sent/received. CSPN finally generates the Petri net graphic specification and an SPNP Petri net specification file named "<fn>_snpn.c" (fn is the name of the input file). All of this process occurs at various levels of user controllable interaction as will be described. In essence the approach provides for systematic and automatic translation and subsequent augmentation (e.g., failure rates, service rates, and deadlines) of the resultant Petri nets for assessing different candidate implementations; –formally (as provided by the grammar) relating stochastic parameters back to the specification level; –and analyzing the stochastic Petri nets using the SPNP tool [10-13, 14, 34].

The CSP-to-Petri net (CSPN) tool is textual based. The initial specification and parameterization work must be completed by using a text editor (see Figures 13 and 14 for examples of P-CSP specifications). Viewing the Petri net's distribution of places and transitions as a graph after a translation is accomplished by setting the "-d" (for *dot*) on the command line.⁸

2.1. Translation phases of the CSPN tool

There are four basic parts involved in the context of Figure 2. First, there is the specification phase. Second, is running CSPN which will invoke any of the available options defined in Table 2. Third is interacting with CSPN (during the run) to control how the SPNP analysis is run and to parameterize the elements of the translation (e.g., set the rates and probabilities which are associated with the resultant transitions). The fourth and last phase is associated with structural

⁸Version 1.0 of CSPN does not automatically invoke the dot program to create the postscript graphic file. To do so use the command: >> *dot -Tps filename.dot > filename.ps*. Dot is available from AT&T Bell Laboratories.

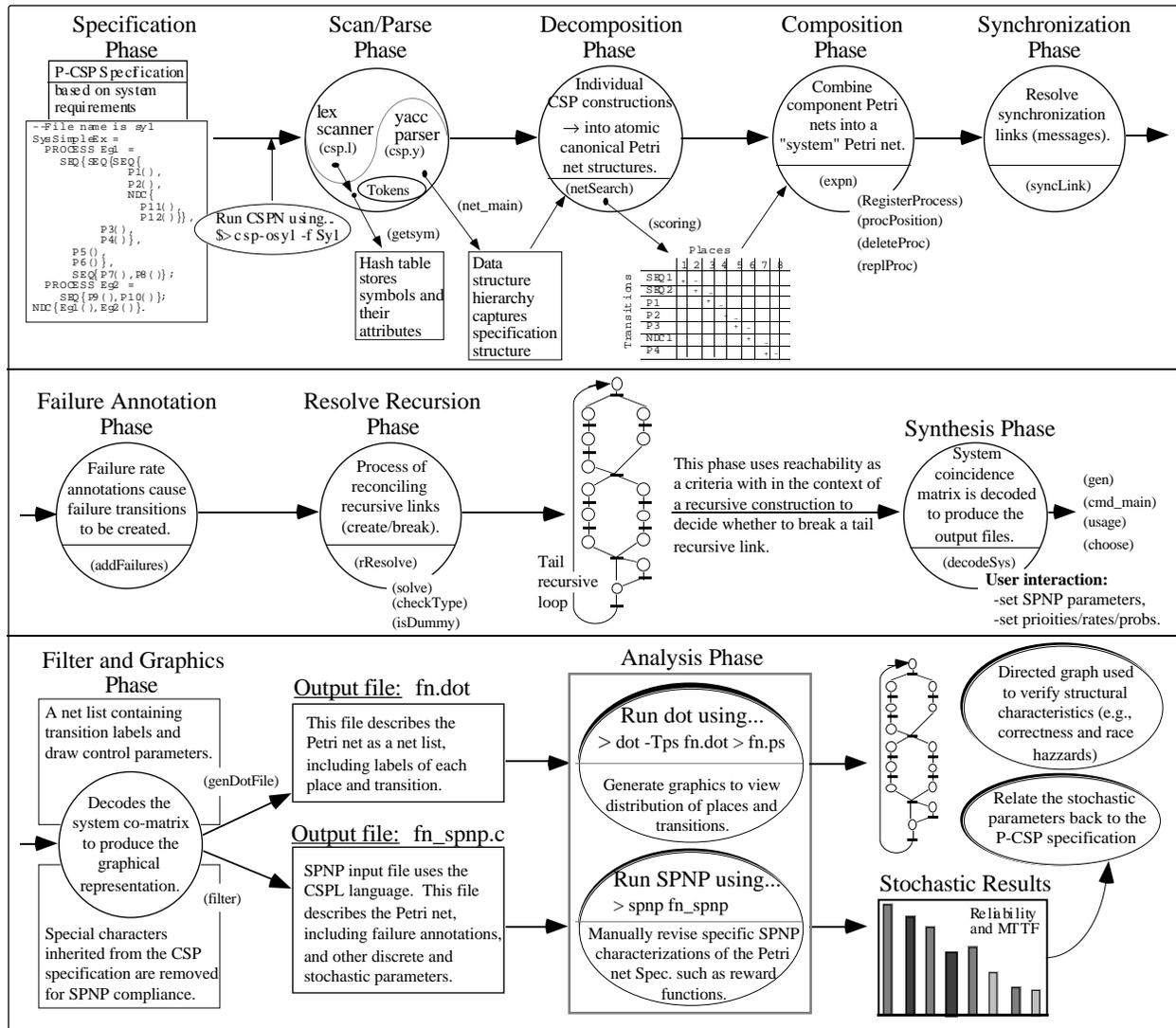


Figure 2. Context diagram and translation phases of the CSPN tool.

and stochastic analysis of the Petri net. Structural analysis involves viewing the distribution of places and transitions of the graphical representation of the Petri net.⁹ The stochastic analysis involves running SPNP to derive dependability and performance results based on the prior phase of parameterizing the model and relating the results to the graph and back to the original specification. The SPNP specification file may be edited to finely tune the characteristics of the SPNP specification prior to running the stochastic analysis.¹⁰ Once SPNP is run, the results can be considered in the process of conducting further analysis.

In viewing Figure 2, note that the following eight steps occur during the translation process: (1) *Scanning and Parsing* (action rules embedded in the parser enable CSPN to capture the

⁹This option causes CSPN to generate a *fn.dot* file which is processed to provide the graphical representation of the Petri net (embedded postscript). The dot tool is used to create the Petri net graphic without manual intervention.

¹⁰The SPNP specification file can be run for a simple analysis without manual intervention.

semantics of the specification), (2) *Decomposition* (allocating/scoring a coincidence matrix for each CSP element and the recording of any annotated service rates and probabilities), (3) *Composition* (combining elemental coincidence matrices and building their requisite process lists), (4) *Synchronization* (resolution or combining of message links), (5) *Failure annotations* (if active, an appropriately annotated process is augmented with a failure transition), (6) *Resolving recursion* (assumes tail recursion and involves finding and linking loop-back arcs as well as breaking tail recursive loops), (7) *Synthesis phase* (takes the system coincidence matrix and creates the SPNP Petri net specification file during an interactive session with the user), and (8) *Filter* (removes special characters inherited from the CSP specification that are not valid in an SPNP specification) and *graphics* (creates a digraph specification net list that is later compiled using dot to produce an embedded postscript graphic). In general, Figure 2 shows the various translation phases and the use of SPNP as it applies to this approach. The names in parenthesis represent the C-function name(s) that are associated with a given phase.

TABLE 2 LISTING OF THE CSPN COMMAND LINE OPTIONS.

Option	Description
-v	Used to set the verbose mode and is only valid when the "-o" option is specified. An interactive menu is invoked which allows the user to set SPNP run parameters.
-f	Used to generate failure transitions into the <i>filename_snp.c</i> file. This option enables detection of failure annotations and causes interactive inputs with the "-o" option specified.
-F	Set to invoke the filter which will replace the 3 special characters (?,!,:) in the <i>filename_snp.c</i> with SPNP compliant characters (_i_, _o_, and _ respectively). Otherwise, SPNP will not compile the input file. Valid only when the "-o" option is used.
-s	Use the default service rates for timed transitions. If no service rate is specified as an annotation then CSPN will use 0.1.
-o<name>	To generate the SPNP input specification file (<i>filename_snp.c</i>) this option <u>must</u> be specified ("name" is optional and the default used is the tool name "cspn").
-i<number>	Number of iterations used by SPNP (default is 2000).
-a<number>	Rate for return to initial marking from absorbing markings (default is 0.0).
-p<number>	Set floating point precision used by SPNP (default is 0.000001).
-P	Set to enable selection of priorities for individual transitions (the default is none).
-d	Set to generate a "dot" graphics file. Dot uses this digraph specification file to generate the graphical representation of the Petri net.
-n	Set to enable a network list file. This file shows how CSPN has interpreted the structural aspects of the CSP specification.
-t	Set to generate a symbol table file. This file will contain all the data recorded for each element (process names, constructions, variables, channels, ...) of the process specification.

2.2. Running the CSPN tool

Running CSPN (i.e., $\$> csp <options> specification-file$) and using the various command line options enables the numerous features and functionalities. For example, if the user is in the process of correcting the syntax of the CSP specification then it would not be necessary to specify any of these options, only the input file. Also, if the user just wants to understand how the CSP specification looks in terms of the structural characteristics (i.e., investigating inherent weaknesses in communications, race hazards etc.) then adding the "-d" option would enable only the production of the graph. The "-F" option invokes a filter and is necessary only when the user plans to run an SPNP analysis. The "-f" option is a nice feature because it enables the analyst to assume a failure free environment by simply ignoring any embedded fail annotations that may exist in the CSP specification (without "-f" CSPN *ignore* failure annotations). Omitting failure annotations from the P-CSP specification has the same affect. The option "-s" streamlines the process of generating the SPNP input specification by assigning default service rates to timed transitions without querying the user to provide such. As mentioned above, the "-o" option generates a file for SPNP analysis. Its best if a file name be given with this option (i.e., "-ofilename"). This settles the problem of overwriting previous files generated using the default name that is assigned by CSPN when no name is provided. The "-i", "-a" and "-p" options are used to parameterize the SPNP run by setting the iteration number, absorbing rate (for recycling back to the initial marking), and precision for floating point operations respectively. The "-P" option is only valid when "-o" is used and enables the user to assign priorities to any of the transitions. The "-d", "-n" and "-t" options are useful when something unexpected happens after running CSPN such as a run time error. The user may wish to rerun the translation and view the internal data structures that are generated during the translation process.

2.3. CSPN data structures

Internally, there are four basic data structures employed by CSPN: (1) *Symbol table* which maintains attributes assigned to all system elements (actions, processes, communications and constructions), (2) *Process lists* which consist of all the names of the associated actions/processes involved in a particular construction, (3) A *network* of linked lists which capture the structure of the specification (adjacency and nesting), and (4) Coincidence matrix which maintains the graphical distributions of places, transitions and their connectivity.

The coincidence matrix (or co-matrix) is the Petri net. Each element and composition from the CSP specification has a co-matrix maintained in the symbol table. The construction of the n-by-m co-matrix is defined in terms of the transitions (i.e., CSP-process names become transition names). Transitions are associated with rows (from top to bottom). Places are associated with columns (numbered from left to right starting from zero). A non-zero element in the matrix A represents an arc which links a transition to a place. Elements (a_{ij}) can have one of three values

(zero, +1 or -1): $a_{ij} = +1$ indicates an arc *from* the transition of row i *to* the place of column j ; $a_{ij} = -1$ indicates an arc *to* the transition of row i *from* the place of column j . The process list stores the transition names in their proper order. The numbering of places is ordered, but their semantics are defined in terms of the transitions.

3. The CSP-based language (P-CSP) primitives

Systems are built from processes. The simplest process is an action (an assignment, input or output). SKIP and STOP are two special processes: they both perform no action (i.e., engage in no event), but SKIP terminates while STOP does not terminate (engages in infinite internal actions) causing a deadlock. Larger processes are built by combining smaller processes. PAR (or ||), SEQ (or ;), NDC (or \sqcap), DC (or \sqcup), and $\text{MU}.x\{\}$ (or $\mu X. P$) are the constructors that can be used for this purpose. The complete CSP-based grammar is provided as a yacc (viz., BNF) specification in [36, 37].¹¹

An example declarative construction would be: `PROCESS My_example = SEQ{P(), Q(), R()};` where the process "My_example" is declared as a sequential composition of three process calls (P, Q, R). A process call is one of the main elements used to denote processes. Processes can have internal structures which are defined in a PROCESS declaration and used later much like a function call in a typical programming language (e.g., `My_example()`). The simple rule to remember is that when a process call is made inside a declarative construction (i.e., between "PROCESS =" and ";") it need not be pre-declared (i.e., the process can be defined on-the-fly rather than being pre-declared).¹² However, when a process is pre-declared it must subsequently be used as a "process call." In the example below, `P1()` and all the other elements in similar format, having "()" appended to their name, are process calls. Process calls in the main body of the specification (last line of this example) *must* be pre-declared:

```
Example_System =
    PROCESS My_example = SEQ{P1(), P2(), P3()};
    PROCESS Your_example = PAR{Q1(), Q2(), My_example()};
    PROCESS Monitor = NDC{SKIP, R1, R2()};
Mu.X{ NDC{ My_example(), Your_example()}}. --dashes are prefixed to comments
```

Pre-declaring a process is a way of abstracting away the internal details of the process function. Translation of a pre-declared process call expands all of its internal structure in

¹¹In P-CSP, process and channel names are capitalized (at least the first letter) while other elements (i.e., actions or messages) use only lower case. These are style guidelines and are not enforced by the CSPN tool.

¹²In the sequence of declarations, a process call defined on the fly can be reused, but not later be declared. Another important syntactic rule is enforced for messages during the translation. Each message variable specified in a synchronized PAR must have a matching input and output (i.e., `channel!messageX` must match `channel?messageX`).

creating a "sub-Petri net." In this way, larger processes are formed from the composition of smaller processes. A statement list is a sequential list of $n \geq 1$ statement(s). A statement can be an event (or trigger) which causes a process to engage in an action (e.g., $a \rightarrow P$). This process is defined as an implication. Input and output of messages require a channel. Channels provide unbuffered, unidirectional point-to-point communication of values between two concurrent processes (similar to Ada rendezvous). A guarded process combines one or more processes, each of which is conditional on an input, a boolean expression or both. An expression can be integer, boolean or relational (boolean expressions must consist of boolean variables prefixed with "@"). Operands can be integers, variables, integer expressions or relational expressions (distinct from boolean). A partial BNF specification of the P-CSP grammar can be found in the Appendix.

3.1. Stochastic Petri nets

The Petri net in its simplest form is a directed bipartite graph, where the two types of nodes are known as *places* (circles) and *transitions* (bars). In this approach, places represent *events* while transitions represent actions.¹³ Other researchers have based their system models on conditions and events (where their events are similar to P-CSP's actions or processes).¹⁴ However, in this approach, modeling is based on the notion in CSP of event-action pairings.

A Stochastic Petri net (SPN) is simply a Petri net which has been extended to permit stochastic analysis. These extensions embed the model into a timed environment by associating a time to each of the transitions in the net. The most general extensions allow the usage of stochastic times (rates) and probabilities.¹⁵ The underlying stochastic process is captured by the "extended reachability graph" (ERG), a reachability graph with additional stochastic information on the arcs. The ERG has been shown to be reducible to a Continuous Time Markov Chain (CTMC) provided that the exponential distribution is associated with the Petri net transitions. Since a SPN permits a probability distribution to be associated with transitions (to express delays or failure rates) they are very suitable for modeling system performance and reliability.¹⁶ Stochastic Petri net markings correspond to the states of an equivalent stochastic process. The transition rate from state M_i to M_j (of the equivalent stochastic process) is given by $q_{ij} = \lambda_{i1} + \lambda_{i2} + \dots + \lambda_{im}$ where λ_{ik} is the delay in firing a transition t_k which takes the Petri net from

¹³CSP processes perform the systems actions, while the events that trigger such actions are characterized by the completion of an action (i.e., process) or the occurrence of conditions that enable the actions (or processes).

¹⁴Murata, describes a slightly different abstraction that defines conditions and events. Murata uses places to represent conditions, and transitions to represent events. A transition has a certain number of input places and output places representing the preconditions and post-conditions of an event (see [Murata89] page 542).

¹⁵When there are multiple transitions enabled by one token, a probability is associated with each of the involved transitions. Such a transition is an immediate and its firing is instantaneous (no time is consumed).

¹⁶Each transition is associated with a random variable that expresses the delay from the enabling to the firing of the transition. When multiple transitions are enabled, the transition with a minimum delay fires first. When the random variable is exponential, the markings of the SPN are isomorphic to the states of a CTMC.

marking M_i to M_j (when several transitions enable the firing from M_i to M_j). See [1, 14, 21-22, 26, 28, 38] for more details on Petri nets, SPNs, Markov and Markov Reward processes.

3.2. SPNP and the C-based Stochastic Petri net Language (CSPL)

The SPNP package allows the user to perform steady state, transient, cumulative transient, and sensitivity analysis of SRNs [42]. The language used for describing stochastic Petri nets for SPNP (Stochastic Petri Net Package) is CSPL. It is a super set of the C language and provides the full expressive power of C. Predefined functions are available to define SPNP objects. A single CSPL file is sufficient to describe any legal SRN because the SPNP user can input (at run-time) the number of places and transitions, the arcs among them, and any other required parameter. The numerical parameters used in the specification of rates and probabilities are incorporated in the same CSPL file. An example of the CSPL file structure is shown in Figure 3.

The function *parameters* allows the user to customize how the package will perform the analysis by setting specific call parameters in the sub-functions *iopt()* and *fopt()*. Several parameters establishing a specific behavior can be selected [11]. The function *net* permits the user to completely define the structure and parameters of an SRN model. The basic functions that can be used inside the net include *place()* for naming all the places, *trans()* for naming all the transitions, *iarc()* for defining a transition's input arc, *oarc()* for the output arcs, and *init()* which defines the initial marking. Probabilistic behavior may be specified using *probval()*, the timing of events can be specified by assigning rates to the transitions in *rateval()*. More advanced functions include *harc()* for making inhibitor arcs while the functions *miarc()*, *moarc()*, and *mharc()* define multiple cardinality input, output

```

parameters(){
    iopt(IOP_PR_MARK_ORDER, VAL_CANONIC);
    iopt(IOP_PR_MERG_MARK, VAL_YES);
    iopt(IOP_PR_FULL_MARK, VAL_NO);
    iopt(IOP_PR_RSET, VAL_NO);
    iopt(IOP_PR_RGRAPH, VAL_NO);
    iopt(IOP_PR_MC, VAL_NO);
    iopt(IOP_PR_MC_ORDER, VAL_FROMTO);
    iopt(IOP_PR_PROB, VAL_NO);
    iopt(IOP_MC, VAL_CTM);
    iopt(IOP_OK_ABSMARK, VAL_NO);
    iopt(IOP_OK_VANLOOP, VAL_NO);
    iopt(IOP_OK_TRANS_M0, VAL_YES);
    iopt(IOP_METHOD, VAL_SSOR);
    iopt(IOP_CUMULATIVE, VAL_YES);
    iopt(IOP_SENSITIVITY, VAL_NO);
    iopt(IOP_ITERATIONS, 2000);
    iopt(IOP_DEBUG, VAL_NO);
    iopt(IOP_USERNAME, VAL_NO);
    fopt(FOP_ABS_RET_M0, 0.000000);
    fopt(FOP_PRECISION, 0.000001);
}
net(){
    /* Definition of places */
    place("p0");
    init("p0",1);
    place("p1");
    place("p2"); ...

    /* Definition of transitions */
    trans("dt1");
    trans("InTransit");
    trans("Togate_o_arrive");
    trans("dt_arrive"); ...

    /* Definition of rates */
    probval("dt1",1.0);
    rateval("InTransit",20.00000000);
    rateval("Togate_o_arrive",19.00000000);
    probval("dt_arrive",1.0);
    rateval("AtIntersection",18.00000000); ...

    /* Definition of input arcs */
    iarc("dt1", "p0");
    iarc("InTransit", "p1");
    iarc("Togate_o_arrive", "p2"); ...

    /* Definition of output arcs */
    oarc("dt1", "p1");
    oarc("dt1", "p8");
    oarc("InTransit", "p2"); ...

}
assert() {
    return(RES_NOERR);
}
ac_init() {
    fprintf(stderr, "\n<<<Run title goes here>>>");
    fprintf(stderr, "\nGenerating SRN data ... \n\n");
    pr_net_info();
}
ac_reach() {
    fprintf(stderr, "\nThe reachability graph is being ");
    fprintf(stderr, "generated ... \n\n");
    pr_rg_info();
}
/* - reward_type definitions go here -----*/
ac_final(){
    int i;
    time value( 0.1 );
    pr_mc_info();
    pr_std_average();
    pr_std_cum_average();
}

```

Figure 3. SPNP input file structure.

and inhibitor arcs (these more advanced functions are not synthesized by CSPN during the translation process).¹⁷

3.3. Mapping CSP to Petri nets using canonical translation rules

An initial set of rules for translating CSP specifications into Petri nets (Petri nets) is defined in [19, 22, 29-30]. The translations between CSP and Petri nets are based on the fact that in CSP, processes execute actions which in turn may enable other actions and in this way CSP processes move from one action to another. Activities which enable a process to be activated can be viewed as conditions (or events) which are represented by places, while the actions themselves are viewed as transitions. Some example translations are shown in Figure 4 (under each of the Petri net constructions is the P-CSP specification).

$a \sqcap b$	$\mu X.(b \sqcap c \rightarrow X)$	$a \parallel b$	$(a \parallel b)$	$(a \sqcap b) \parallel_{(a,b)} (a \parallel b)$	$(a \rightarrow b \rightarrow c) \parallel_{(b)} (d \rightarrow b \rightarrow e)$
NDC{ a, b }	$\text{Mu.X}\{\text{NDC}\{b, c \to X\}\}$	PAR{ a, b }	DC{ a AND {ch1 ? msg1}, b AND {ch2 ? msg2} }	PAR{ NDC{a, b}, DC{a AND {ch1 ? msg1}, b AND {ch2 ? msg2} } (a,b) }	PAR{ {a→b→c}, {d→b→e}(b) }
A. Nondeterministic choice to proc a or b	B. Nondeterministic choice w/ recursion	C. Parallel actions are transitions	D. Deterministic choice	E. Non- and deterministic choices run in parallel	F. Parallel actions synchronize on b

Figure 4. Example of some representative CSP \rightarrow Petri net translation rules (see next figure).

The CSP to Petri net translations were designed to facilitate the automatic decomposition of the CSP constructs into Petri net sub-components and subsequent re-composition of the subnet components into a complete *system* Petri net. The Petri net translation from a given CSP construction (i.e., specification) need not be unique because ultimately, when we combine the subnets, we must introduce dummy places and dummy transitions to maintain the complete Petri net's bipartite nature.¹⁸ And, once the complete *system* net is obtained, the structure itself may

¹⁷Guards (logical conditioning functions associated with a transition) and priorities can be specified using guard() and priority(). Marking dependence is specifiable using mark() and enabled().

¹⁸Intuitively, it is possible to reduce different Petri net equivalents into a canonical form. A set of canonical translation rules are applied to derive each component's Petri net equivalent [37].

be reduced (e.g., by combining adjacent dummy transitions or collapsing such places and transitions into their predecessor/successor transitions) to something that is trace equivalent to the CSP specification. This in itself is all that is necessary to define a complete set of markings and hence an equivalent Markov process.[38]¹⁹

Petri nets are inherently non-deterministic and asynchronous while CSP is inherently deterministic and synchronous (though an explicit definition of non-deterministic choice exists in CSP). Since the purpose is stochastic analysis, we depend on the non-deterministic nature of the Petri nets to conduct the stochastic analysis. This implies that the determinism of CSP is also translated into the non-determinism of Petri nets. However, the resolution of structure within the translation is standardized and deterministic, while the probabilistic transitioning is only applied to the appropriate nondeterministic choice composition operator of CSP. In any event, the goal is to demonstrate the feasibility of translating from CSP and Petri nets by decomposing a CSP specification into its component parts (processes, channels, constructors etc.) and this is done by choosing one standard (canonical) translation path from among equivalents. Section 4 provides some mechanisms and conventions which have been defined for the canonical translations.

4. Some CSPN mechanics

This section provides some of the CSPN implementation details. The canonical translation rules are codified in CSPN. The set of details provided give the basic framework for CSPN.

4.1. Co-matrix expansion

When the P-CSP specification is parsed, each construct (e.g., PAR, SEQ, etc.) of the specification is separated into its component elements (process names, channels, variables) and represented as a sub-Petri net. The sequential construction shown in Figure 5 illustrates how the co-matrix is used to represent, in this case a SEQ composition as a Petri net. Figure 6 shows the same translation for the parallel (i.e., PAR) construct.

Combining the component co-matrices to produce a complete system Petri net is a process of co-matrix expansion. The expansion is constrained in two dimensions to preserve the algebraic structure associated with (1) adjacency and (2) nesting. Figures 7 and 8 give an illustration of the expansion process.

The test shown in Figure 7 is designed to determine, based on the location of the transition to be expanded, which method of expansion to use. In Figure 8, an expansion is performed using method 3. The co-matrix SEQ0 (analogous to co-matrix A in Figure 7) is expanded by inserting SEQ1. SEQ1 consists of two processes, P1 and P2 (and is analogous to co-matrix B in Figure 7). The expansion must replace the transition SEQ1 by the two process names P1 and P2. The final *combined* result retains the SEQ0 name.

¹⁹A task which is left to the Petri net tool (i.e., SPNP).

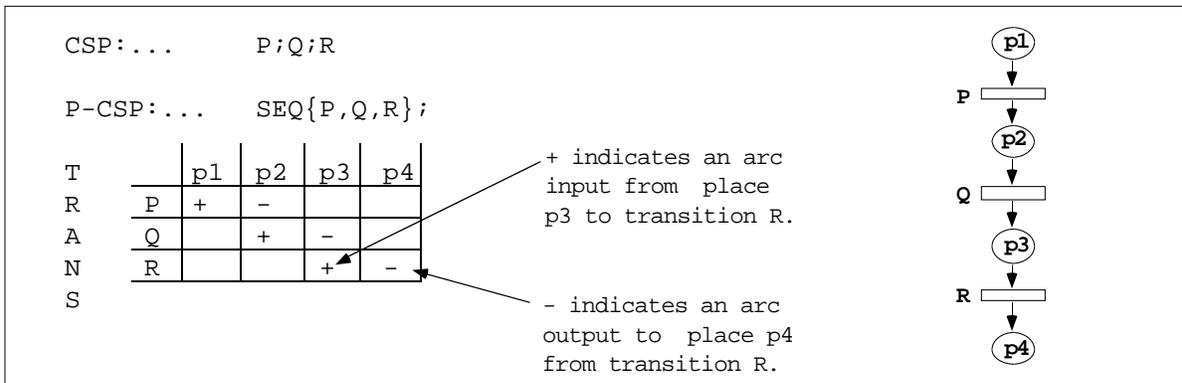


Figure 5. SEQ construct with co-matrix and Petri net representations.

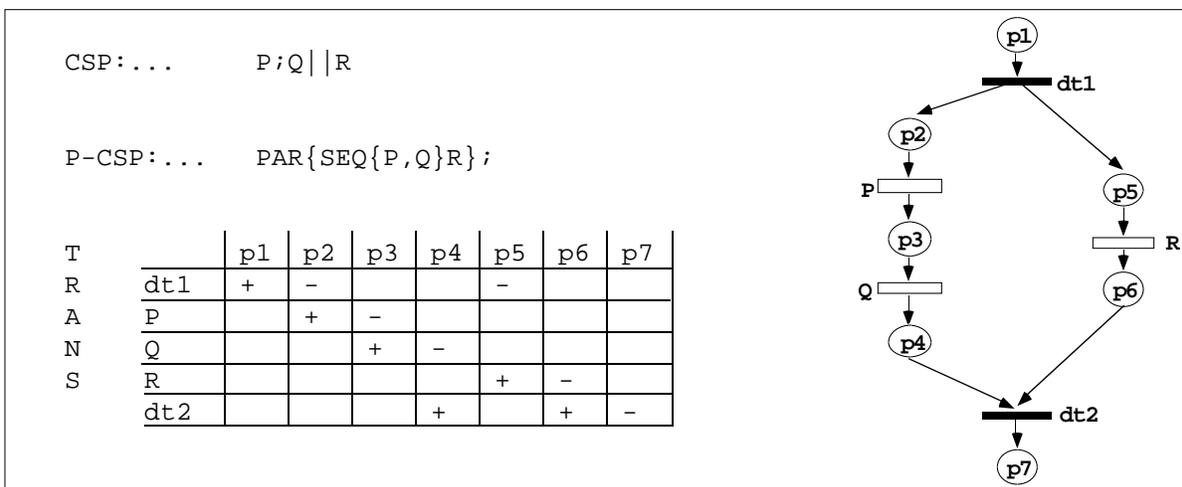


Figure 6. PAR and SEQ construct with co-matrix and Petri net representations.

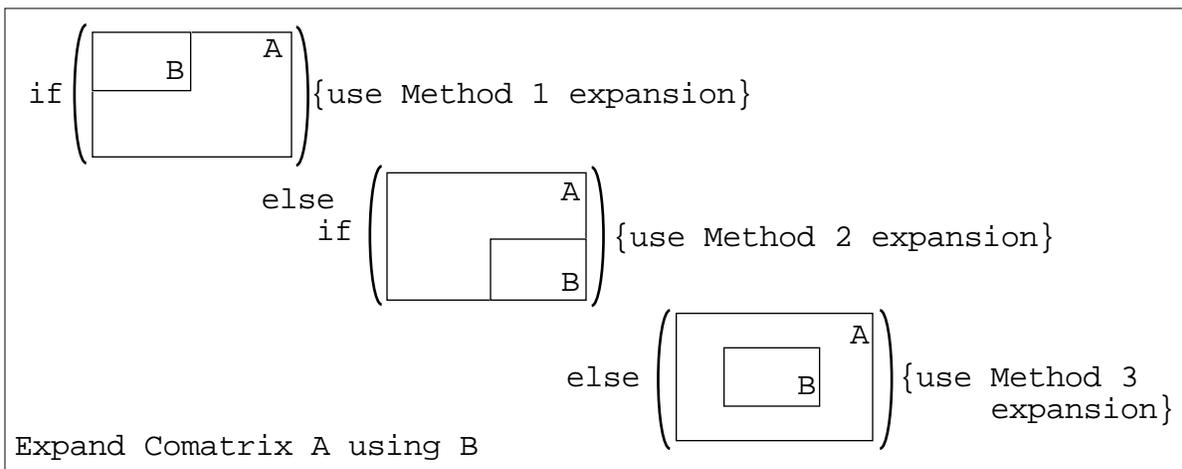


Figure 7. Choosing a combining method 1, 2 or 3 for expansion depends on locality.

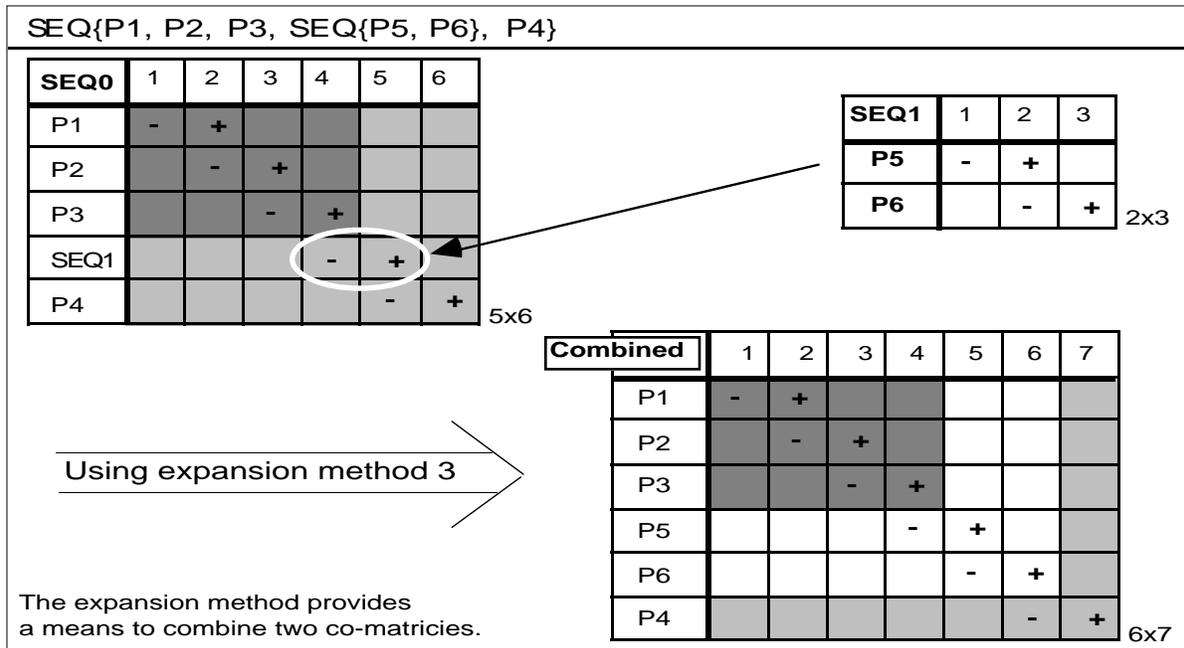


Figure 8. Identifying the transition to expand into a larger Petri net.

Note, the term SEQ is a key word (used for sequential composition of processes), and is considered itself to be a process. CSPN treats each occurrence of this type as a unique process by appending a unique value to the name (e.g., 0 is appended to the first occurrence of SEQ to give SEQ0 and the next occurrence of SEQ will have "1" appended). In this way, CSPN tracks each occurrence of a given type of keyword (i.e., SEQ, PAR, NDC, DC, STOP and SKIP).

4.2. CSP represented as a network structure

A network of linked lists is used to capture the algebraic structure in two dimensions (1) adjacency (among declared processes or within a process) and (2) nesting within processes. Two examples are provided in Figure 9 and 10. The first one emphasizes adjacency and the other emphasizes nesting.

There are two main network arrays used (1) SYS[], each SYS[i] points to a process defined in a PROCESS declaration and (2) NET each NET[i] points to a "process" element as defined by the P-CSP grammar (e.g., process call, constructor like PAR, NDC, etc.). Figure 11 gives some details of the three major C data structures employed by CSPN: (1) Symbol table entry (as described above) as well as (2) NET_NODE and (3) NODE structures which together provide the building blocks for the network.

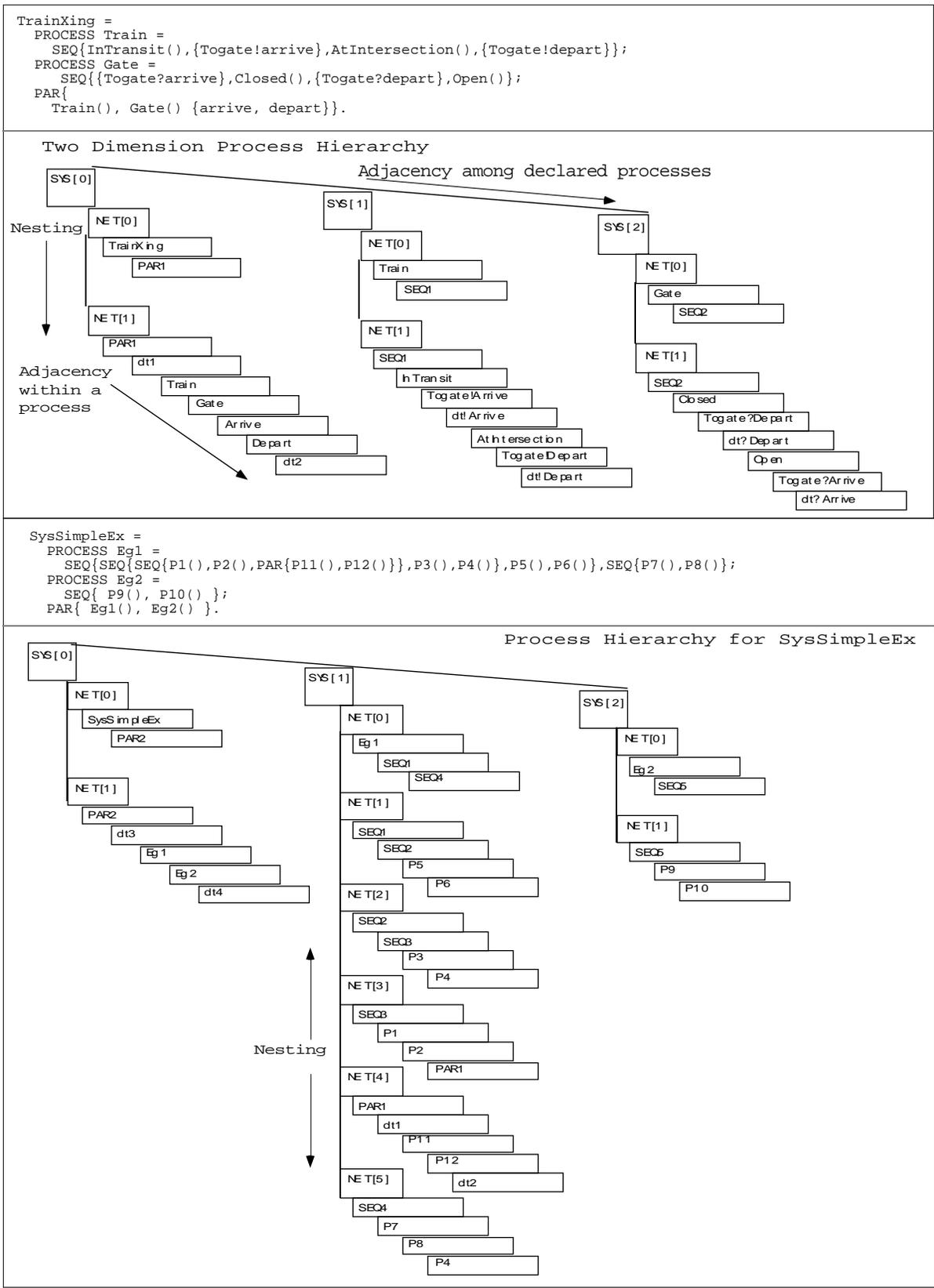


Figure 10. Process hierarchy for system "SysSimpleEx" with exaggerated nesting.

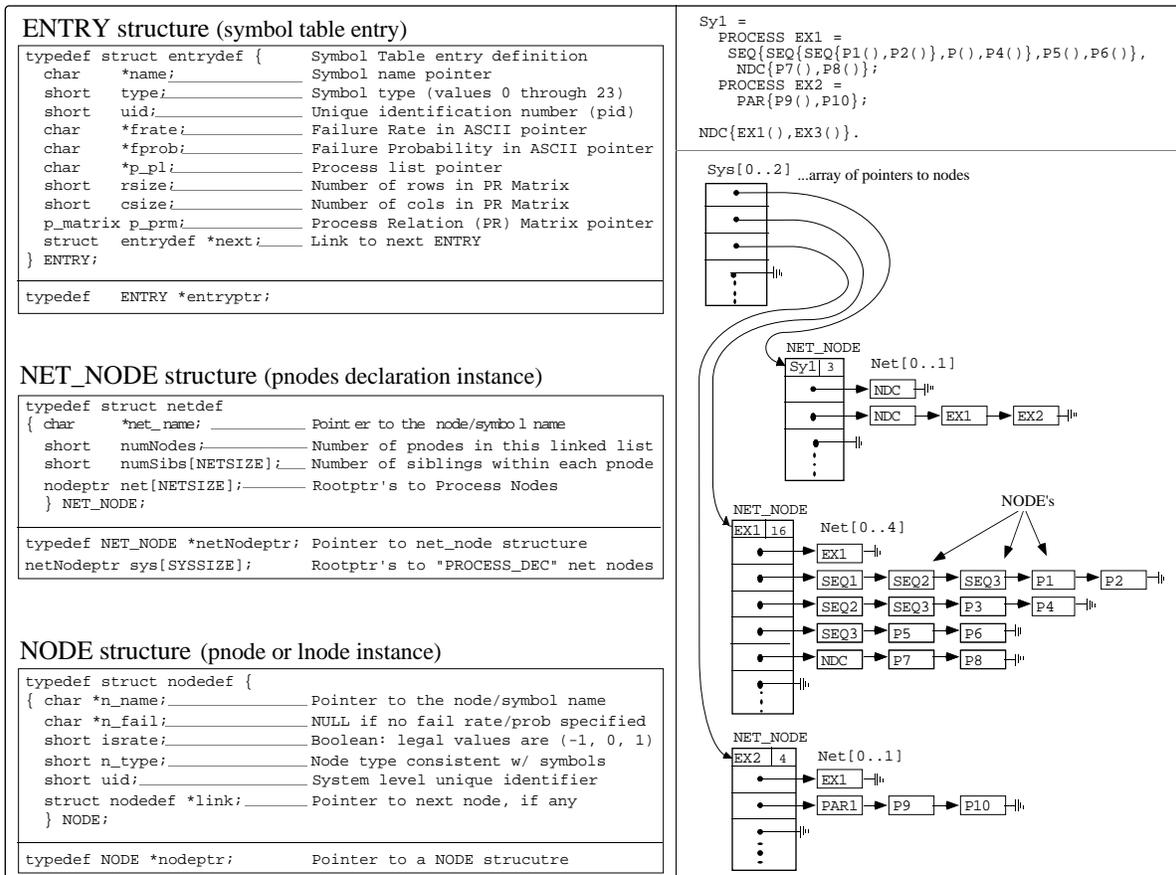


Figure 11. Entry, net_node, and node data structures.

5. Summary

The objective in this work was to show that CSP specifications can be translated into SPNs for the purpose of reliability and performance analyses. Such translations can give insight (1) into the feasibility of meeting non-functional requirements, (2) by helping to identify the best candidate design based on a formal description of the system, (3) by helping to identify failure modes and fault handling mechanisms. This approach enables the stochastic properties of the system specification to be ascertained while allowing the parameters used in the analysis to be formally captured in the P-CSP design specification. Subsequent analyses can be run without having to rewrite all of the pertinent values. Only those parameters that are identified as critical in terms of their impact to the integrity of the overall system (i.e., sensitivity analysis) need be perturbed. This approach provides feedback to the designer so that a judicious cost-benefit analysis in terms of fault-avoidance and fault-tolerance can be made.

A textual language for CSP specifications was designed. A tool was implemented for translating the CSP specifications into stochastic Petri nets. The Petri nets are coded in the form of a coincidence matrix. The graphical representation of the resulting Petri net can be viewed

using tools such as dot.²⁰ The coincidence matrix is then converted into the format needed for analysis using SPNP. The tool has been tested using a diverse set of process compositions (see Figures 13 and 14 for some example test cases). A simple communication protocol from [39, page 253] is shown in Figure 14C. Except for dummy transitions and places which are the artifacts of the canonical translation rules, the Petri nets generated by CSPN can be compared to the original Petri nets from which the P-CSP was originally derived.

The tool combines the power of two other tools namely dot (for viewing the graphical PN representation) and SPNP (Stochastic Petri Net Package for stochastic analysis). The CSPN interface is textual. CSPN offers a rich selection of command line options. Most of CSPN's current features are driven by the SPNP functionalities. An interactive menu is one option that is used to control run parameters related to the type of analysis (e.g., precision, iterations, generating a reachability graph, running continuous time versus discrete time Markov analysis, etc.). Another menu allows the designer to parameterize and control the character of the system under study (e.g., setting priorities, rates or probabilities among transitions, etc.). In general, the CSPN tool provides a new level of abstraction and basis for understanding interactive concurrent process algebraic specifications by leveraging the power of dot and SPNP.

5.1 Future enhancements

This work can be extended to incorporate a broader scope of translations and the characterization of properties other than structural that are useful for error avoidance, fault tolerance, detection of deadlocks, unsafe behaviors, and timeliness. Limitations include (1) ease of use (e.g., GUI), (2) devising a mechanisms to eliminate extra and adjacent dummy transitions, (3) expanding the language to incorporate some of the ideas of real-time CSP, (4) defining a new extended grammar based on other formal specification languages, and (5) validating the approach by applying the method to larger examples and/or a real system.

In the future, the critical elements of a formal requirement specification will be abstracted into a formal P-CSP design specification. CSPN will be used to translate the model into stochastic Petri nets. Subsequent analysis of the specification's structural characteristics will reveal the system and component reliability (based on empirical data and/or the postulated failure rates) of such components. The system model will then be characterized in terms of the ascertained failure behaviors. Subsequent models will be perturbed and re-evaluated for comparison purpose.

5.2 CSPN specifications²¹

CSPN version 1.0 is currently 8,500 lines of C code running on a Unix-based Sun 4.

²⁰See Drawing graphs with dot by Eleftherios Koutsoufios and Stephen C. North at AT&T Bell Laboratories.

²¹**Acknowledgement:** Thanks to Krishna Kavi for some critical comments he made on an earlier draft of this paper.

6. References

1. Balbo, Gianfranco, "On the Success of Stochastic Petri Nets," IEEE Proceedings Petri Nets and Performance Modeling 95, Durham, NC, pp. 2-9, October 3-6, 1995.
2. Balbo, Gianfranco, Donatelli, S., Franceschinis, G., Mazzeo, A., Mazzocca, N. and Ribaud, M., "On the Computation of Performance Characteristics of Concurrent Programs Using GSPNs," Performance Evaluation, Vol. 19, pp. 195-222, 1994.
3. Bernardo, Marco, Busi, Nadia and Gorrieri, Roberto, "A Distributed Semantics for EMPA Based on Stochastic Contextual Nets," The Computer Journal, Vol. 38, No. 3, 1995.
4. Bernardo, Marco, Donatiello, Lorenzo and Gorrieri, Roberto, "Giving a Net Semantics to Markovian Process Algebra," IEEE Proceedings of the 5th International Workshop on Petri Nets and Performance Modeling, Durham, NC, pp. 169-178, October 3-6, 1995.
5. Butler, Ricky and Johnson, Sally C., "Formal Methods for Life-Critical Software," Proceedings of the AIAA Computing in Aerospace 9, pp. 319-329, Oct. 19-21, 1993.
6. Chiola, Giovanni, Dutheillet, Claude, Franceschinis, Giuliana and Haddad, Serge, "Stochastic Well-Formed Colored Nets and symmetric Modeling Applications," IEEE Transactions on Computers, Vol. 42, No. 11, pp. 1343-1360, November 1993.
7. Chiola, Giovanni, Marsan, Marco Ajmone, Balbo, Gianfranco and Conte, Gianni, "Generalized Stochastic Petri Nets: A Definition at the Net Level and Its Implications," IEEE Transactions on Software Engineering, Vol. 19, No. 2, pp. 89-107, February 1993.
8. Ciardo, Gianfranco and Muppala, Jogesh K., "Manual for the SPNP Package Version 3.1," EE Department, Duke University, Durham, NC, 34 pages, October 18, 1992.
9. Ciardo, Gianfranco and Trivedi, Kishor S., "A Decomposition Approach for Stochastic Petri Net Models," Proceedings of the Fourth International Workshop of Petri Nets and Performance Models, IEEE CS Press, Los Alamitos, CA, pp. 74-83, December 1991.
10. Ciardo, Gianfranco and Trivedi, Kishor S., "A Decomposition Approach for Stochastic Petri Net Models," Proceedings of the Fourth International Workshop of Petri Nets and Performance Models, IEEE CS Press, Los Alamitos, CA, pp. 74-83, December 1991.
11. Ciardo, Gianfranco and Trivedi, Kishor S., "SPNP: The Stochastic Petri Net Package (Ver. 3.1)," MASCOTS'93 Simulation Series, Vol. 25, No. 1, 1993, pp. 390-391, January 17-20.
12. Ciardo, Gianfranco, "Toward a Definition of Modeling Power for Stochastic Petri Net Models," International Workshop on Petri Nets and Performance Models, Madison, Wisconsin, pp. 54-62, August 24-26, 1987.
13. Ciardo, Gianfranco, Muppala, Jogesh and Trivedi, Kishor S., "On the Solution of GSPN Reward Models," Performance Evaluation, Vol. 12, pp. 237-253, December 1991.
14. Ciardo, Gianfranco, Muppala, Jogesh K. and Trivedi, Kishor, "SPNP: Stochastic Petri Net Package," International Workshop on Petri Nets and Performance Models, Kyoto, Japan, pp. 142-151, December 11-13, 1989.
15. Davies, Jim and Schneider, Steve, "Real-Time CSP," Theories and Experiences for Real-Time System Development, AMAST Series in Computing: Vol. 2, Teodor Rus and Charles Rattray Eds., World Scientific, New Jersey, pp. 31-82, 1994.
16. Dillon, L.K., Kutty, G., Moser, L.E., Melliar-Smith, P.M. and Ramakrishna, Y.S., "Graphical Specifications for Concurrent Software Systems," International Conference on Software Engineering, Melbourne Australia, 11 pages, May 1992.
17. Donatelli, S., Ribaud, M. and Hillston, J., "A Comparison of Performance Evaluation Process Algebra and Generalized Stochastic Petri Nets," IEEE Proceedings, Petri Net and Performance Modeling (PNPM), Durham, NC, pp. 158-168, October 3-6, 1995.
18. Donatelli, Susanna, Franceschinis, Giuliana, Mazzocca, Nicola and Russo, Stefano, "Software Architecture of the EPOCA Integrated Environment," Proc. 7th Int'l Conf. on Computer Performance Evaluation Modeling Techniques and Tools, LNCS 794, G. Goos and J. Hartmanis Eds., Springer-Verlag, Vienna, Austria, pp. 335-352, May 1994.
19. Gerhart, Susan L., "Applications of Formal Methods: Developing Virtuoso Software," IEEE Software, pp. 7-10, September 1990.

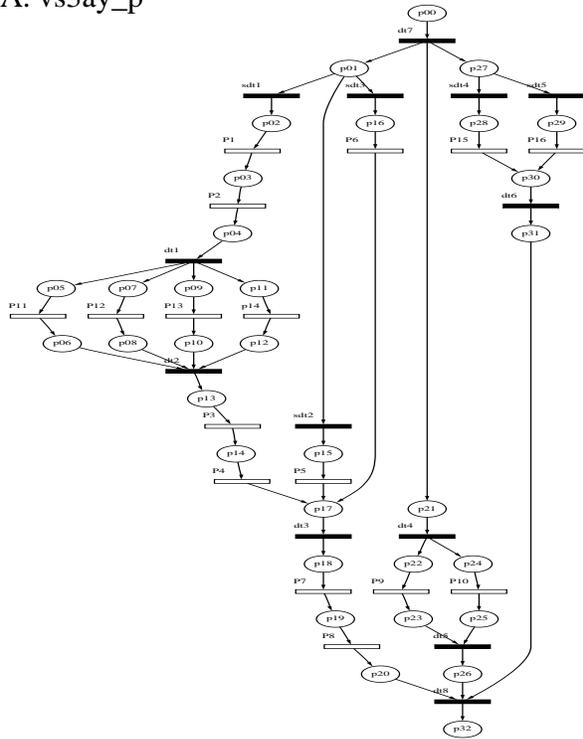
20. Hoare, C.A.R., *Communicating Sequential Processes*, Prentice- Hall International Series in Computer Science, 256 pages, 1985.
21. Johnson, Barry W., *Design and Analysis of Fault-Tolerant Digital Systems*, Addison-Wesley Publishing Company, 584 pages, 1989.
22. Kavi, K.M. and Buckles, B.P., "Formal Methods for the Specification and Analysis of Concurrent Systems" Tutorial Notes, 1993 International Conference on Parallel Processing, Lake Charles, IL., 75 pages, Aug. 20, 1993.
23. Kavi, K.M., and Sheldon, F.T., "Specification of Stochastic Properties with CSP," IEEE Proc. Int'l Conf. on Parallel and Distributed Systems, Taiwan, pp. 288-293, Dec. 1994.
24. Kavi, K.M., Sheldon, F.T. and Reed, S.C., "Specification and Analysis of Real-Time Systems Using CSP and Petri Nets," Int'l Journal of Software Engineering and Knowledge Engineering –Special Issue on S/E Practices and Tools for Real-Time Systems, June 1996.
25. Kavi, Krishna M. and Sheldon, Frederick T., "Reliability Analysis of CSP Specifications Using Petri Net and Markov Models," HICSS-28, January 1995.
26. Laprie, J.C., Kaaniche, M., and Kanoun, K., "Modeling Computer Systems Evolutions: Non-Stationary Processes and Stochastic Petri Nets –Application to Dependability Growth," IEEE PNPM'95, Durham, NC, pp.221-230, October 1995.
27. Liu, Zhiming and Joseph, Mathai, "Transformation of Programs for Fault-Tolerance," Formal Aspects of Computing, Vol. 4, No. 5, pp. 442-469, 1992.
28. Murata, Tadao, "Petri Nets: Properties, Analysis and Applications," Proceedings of the IEEE, Vol. 77, No. 4, pp. 541-580, April 1989.
29. Olderog, Ernst-Rudiger, "Operational Petri Net Semantics for CCSP," Lecture Notes in Computer Science, Springer-Verlag, Vol. 266, pp. 196-223, 1987.
30. Olderog, Ernst-Rudiger, "TCSP: Theory of Communicating Sequential Processes," Lecture Notes in Computer Science, Springer-Verlag, Vol. 255, pp. 441-465, 1986.
31. Ostroff, Jonathan S., "Formal Methods for the Specification and Design of Real-Time Safety Critical Systems," Journal of Systems Software, Vol. 18, pp. 33-60, 1992.
32. Priami, Corrado, "Integrating Behavioral and Performance Analysis with Topology Information," Proceedings of the PAPM 95, (& The Computer Journal, Dec. 1995), 1995.
33. Reisig, W., "Combining Petri Nets and Other Formal Methods," 13th International Conference on Application and Theory of Petri Nets, pp. 24-44, June 1992.
34. Sahner, Robin A. and Trivedi, Kishor S., "A Software Tool for Learning About Stochastic Models," IEEE Transactions on Education, Vol. 36, No. 1, pp. 56-61, February 1993.
35. Sheldon, F.T., and Kavi, K.M., "Linking Software Failure Behavior to Specification Characteristics II," IEEE Proc. Fourth Int'l Workshop on Evaluation Techniques for Dependable Systems, San Antonio, TX, Oct. 1995.
36. Sheldon, F.T., Kavi, K.M. and Kamangar, F.A., "Reliability Analysis of CSP Specifications: A New Method Using Petri Nets," AIAA Proceedings Computing in Aerospace 10, pp. 317-326, March 1995.
37. Sheldon, F.T., "Specification and Analysis of Stochastic Properties for Concurrent Systems Expressed Using CSP," Ph.D. Dissertation, Comp. Sci. and Engrng. Dept., Univ. of TX at Arlington, May 1996.
38. Sorensen, Erling Vagn, Nordahl, Jens and Hansen, Niels Herman, "From CSP Models to Markov Models," IEEE Trans of SE, Vol. 19, No. 6, 1, pp. 554-570, June 1993.
39. Tanenbaum, A.S., *Computer Networks*, 2nd Ed., Prentice Hall, Englewood Cliffs, 1989.
40. Van Glabbeek, Rob, Smolka, Scott A., Steffen, Bernhard and Tofts, Chris M.N., "Reactive, Generative, and Stratified Models of Probabilistic Processes," IEEE Proceedings Symposium on Logic in Computer Science, pp. 130-141, 1990.
41. Wang, Chang-Yu and Trivedi, Kishor, "Integration of Specification for Modeling and Specification for System Design," Department of Electrical Engineering, Technical Report from Duke University, Durham, NC, 22 refs., 21 pages, 1994.
42. Wang, Chang-Yu, "Some Problems in the Specification and Analysis of Computers and Networks, Ph.D. Dissertation, Department of Computer Science, Duke University, 101 refs., 218 pages, 1995.

7. Appendix

<pre> ** Rules ** 1: system production */ system: Identifier Equals processdeclist processlist1 Dot; ** 2: processdec (way to declare process names) processdec: PROCESS Identifier Equals processlist1 Semicolon; ** 2.5: processdeclist (list multiple decl under system) processdeclist: processdeclist processdec; ** 3: process definition process: STOP SKIP LeftBrace stmtlist RightBrace PAR LeftBrace processlist2 synclist RightBrace SEQ LeftBrace processlist1 RightBrace NDC LeftBrace processlist2 RightBrace DC LeftBrace guardedproclst RightBrace MU Dot LeftBrace processlist1 RightBrace processcall; ** 4: processlist1 processlist1: processlist1 process; ** 4.5: processlist2 processlist2: processlist2 process process; ** 4.7: synclist synclist: LeftParen anyvarlist RightParen; ** 4.8: anyvar anyvar: booleanvar variable; ** 4.9: anyvarlist anyvarlist: anyvar anyvarlist Comma anyvar; ** 5: statement list stmtlist: stmtlist stmt; ** 6: statement stmt: implication process expression input **looks like {channel ? variable} output **looks like {channel ! variable}; ** 6.3: implication (a statement event -> action) implication: stmt Arrow process; ** 6.6: processcall (instance of a declared PROCESS). processcall: Identifier LeftParen RightParen **Symbol lookup ensures identifier was declared ; ** 7: assignment is covered by expression in integer ** 8: input input: channel InSym variable; ** 9: output output: channel OutSym expression; ** 10: guarded process guardedprocess: guard process; ** 11: guarded process list guardedproclst: guardedproclst guardedprocess; </pre>	<pre> ** 12: guard guard: input booleanexpr booleanexpr AND input booleanexpr AND SKIP; ** 13: recursive definition (defined in process) ** 14: channel channel: Identifier; ** 15: variable variable: Identifier; ** 16: boolean variable (AtSym to distinguish 15, 16) booleanvar: AtSym Identifier; ** 17: expression expression: integerexpr booleanexpr relationalexpr; ** 18: boolean expression booleanexpr: booleanvar TRUE FALSE booleanexpr AND booleanexpr booleanexpr OR booleanexpr NOT booleanexpr booleanvar VarAsgn booleanexpr; ** 19: relational expression relationalexpr: operand LESym operand operand LTSym operand operand EQSym operand operand NESym operand operand GESym operand operand GTSym operand; ** 20: integer expression integerexpr: Negative operand operand Plus operand operand Minus operand operand Star operand operand Slash operand operand VarAsgn operand; ** 21: operand operand: Integer variable integerexpr relationalexpr; ** 22: monadic operand (never used) ** 23: dyadic operand (never used) ** 24: integer is defined in lexer ** 25: digits are defined in lexer ** 26: digit is defined in lexer ** 27: declaration (currently undefined) </pre>
---	--

Figure 12. P-CSP grammar.

A. vs3ay_p



```
--vs3ay_p
--Used to test the nesting of NDC
vs3ay_p =

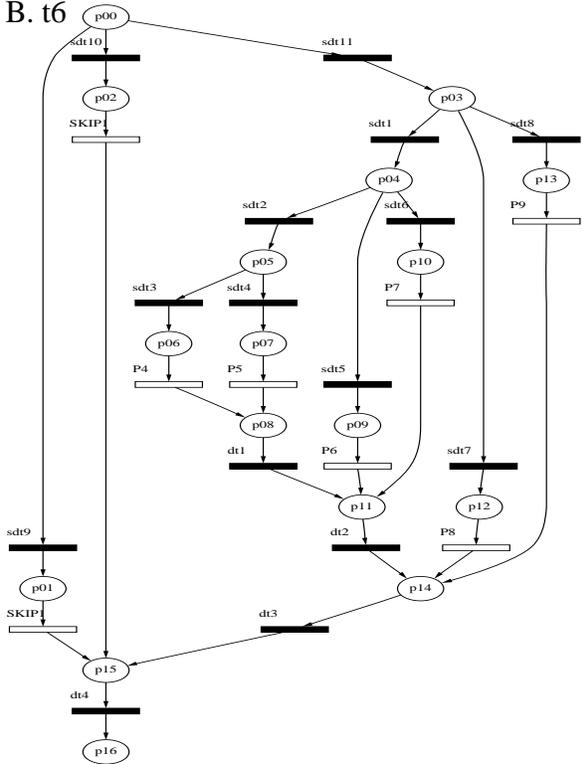
PROCESS P1 = { SKIP };

PROCESS P2 = SEQ{ P1() };

PROCESS P3 = NDC{
  NDC{
    NDC{
      P4(), P5()
    },
    P6(), P7()
  },
  P8(), P9()
};

NDC{
  P1(),
  P2(),
  P3()
}.
```

B. t6



```
--t6
--Example tests NDC, SEQ and PAR nesting and
--non-nested PAR. There are 19 process calls, 3
--process declarations, 3 SEQs, 3 PARs, and 2 NDCs.

SysEg_t6 =

PROCESS Eg1 = NDC{
  SEQ{
    SEQ{
      P1(),
      P2(),
      PAR{
        P11(),
        P12(),
        P13(),
        P14()
      }
    },
    P3(),
    P4()
  },
  P5(),
  P6()
},
  SEQ{
    P7(),
    P8()
  };

PROCESS Eg2 = PAR{
  P9(),
  P10()
};

PROCESS Eg3 = NDC{
  P15(),
  P16()
};

PAR{
  Eg1(),
  Eg2(),
  Eg3()
}.
```

Figure 13. CSPN Petri net translations and their matching P-CSP Specifications (A & B).

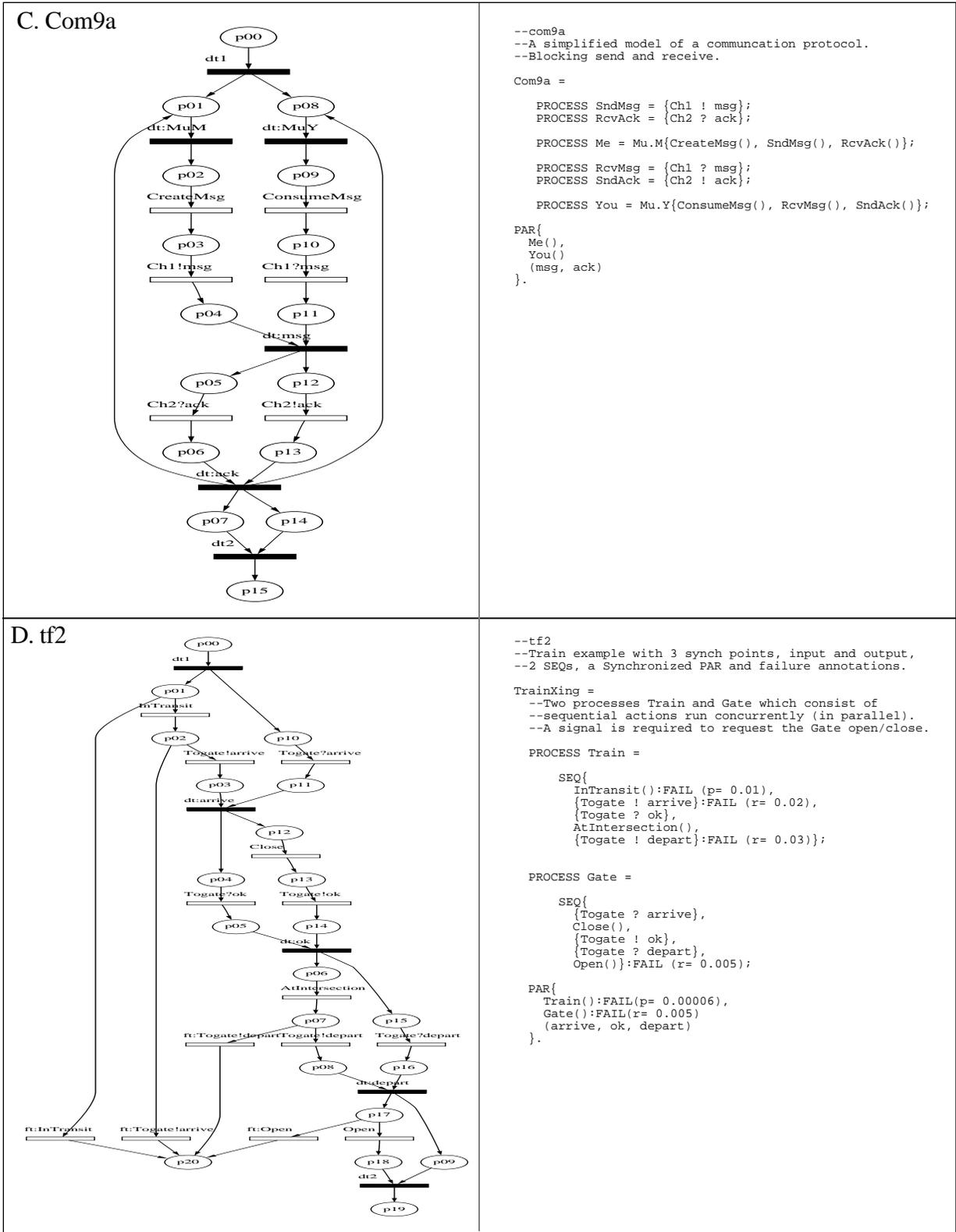


Figure 14. CSPN Petri net translations and their matching P-CSP Specifications (C & D).