Design Patterns in Petri Net System Modeling

Martin Naedele, Jörn W. Janneck
Swiss Federal Institute of Technology Zurich
Computer Engineering and Networks Laboratory (TIK)
CH-8092 Zürich, Switzerland
{naedele,janneck}@tik.ee.ethz.ch

Abstract

Petri nets are an established and well researched means for systems modeling and simulation, but its use in the engineering community is not as widespread as the applicability of the formalism would suggest. A reason for this might lie in the fact that there is no established concept for the concise presentation of reusable Petri net design knowledge. This paper proposes Petri net design patterns as a style of presentation of such design knowledge ranging from building blocks to architectural considerations. The template for description is introduced using a number of examples taken from our design experience.

1. Introduction

Petri nets are an established and well researched model of computation that is applicable in various domains, not only in economics, work flow modeling, and theoretical computer science, but also for the design of complex concurrent computer systems. However, users are mostly found in academic and research environments, whereas the engineering community seems to be reluctant to adopt Petri nets for systems modeling and simulation. A reason for this may be found in the fact that while there exists a fair number of introductory literature on Petri nets as well as a large body of advanced theoretical papers, there are to our knowledge no references available that would teach a practising engineer how to apply the basic concepts of Petri nets to the modeling of systems of greater complexity.

From our experience in introducing Petri nets to design engineers we conclude that it is easy to grasp the basic concepts of places, transitions, and concurrent sequences, and that one is very quickly able to apply this to the modeling of systems that have an internal structure that only consists of applications of those basic concepts. The situation changes when one tries to model complex technical systems. It appears to us that there is a need for a structured collection of easily accessible design experience that can be used as reference for "best practices" by the design engineer who is not so much interested in learning about the theoretical implications of Petri net semantics but rather wants to use the formalism to specify, simulate, and evaluate systems.

The main objective of this paper is to give an initial impulse to collect and spread Petri net modeling knowledge in the form of building blocks and pattern descriptions in order to make Petri nets more popular for the use in real world applications. The representation of design knowledge and experience in the form of patterns is increasingly being used in the software engineering domain (see section 3), and we believe that it can successfully be transferred to the area of Petri net modeling. We intend to provide a starting point for discussion by introducing the concept of Petri net patterns with a small number of patterns taken from our practical experience [15]. The principal issue of this paper, though, is to demonstrate a specific way of illustrating patterns, not the presentation of the patterns we selected as examples. Although we are not aware that any of the patterns in this paper have been described elsewhere, we do not claim that they are original or overly sophisticated. We do, however, know from our experience that they are sufficiently intricate to be considered valuable information in addition to descriptions of the basic Petri net elements.

The paper is organized as follows: Section 2 contains a short reference on the Petri net language we use in the pattern descriptions later on. In section 3 we give an introduction to design patterns and suggest a notational template. The main part of the paper, section 4, presents a detailed description of four Petri net design patterns. We close with a discussion of ideas we have on further work to be done in the area of Petri net design patterns in order to make the Petri nets formalism more useful for engineers.

2. Petri nets

Petri nets [18][19][20] are a graphical and mathematical formalism for modeling, simulation, and formal analysis of

discrete event systems [25]. Petri nets allow the representation of both control and data flow within one formalism. Other frequently used modeling formalisms like finite state machines (FSM), marked graphs, and Synchronous Data Flow are specialisations of Petri nets.

2.1. Basic Petri nets

There exist many different Petri net variants and definitions, which are extensions of so-called condition/event nets:

Definition 1 (Condition/event net) *A condition/event net is defined as a 4-tuple $CEN = (P, T, F, M_0)$, where*

- *P is a finite set of places,*
- *T is a finite set of transitions,*
- *$P \cup T \neq \emptyset$*
- *$P \cap T = \emptyset$*
- *$F \subseteq (P \times T) \cup (T \times P)$ is a set of directed arcs.*
- *$M_0 : P \rightarrow \{0, 1\}$ is the initial marking. It defines the initial number of undistinguishable tokens on each place $p \in P$.*

A Petri net is a bipartite graph, which means that $p \in P$ and $t \in T$ alternate along each path in the graph. In the usual graphical representation of Petri net graphs, places are depicted as circles and transitions as rectangles.

The marking, that is, the distribution of tokens on places, represents the state of a Petri net model. Transition firings change the token distribution and thus the state of the system. They may reflect the occurrence of events or the execution of an operation. The flow of tokens can be defined as follows:

Definition 2 (Enabling rule) *A transition $t_j \in T$ is enabled if all input places of t_j contain a token and all output places are empty, $M(p_i) = 1 \; \forall \; p_i | (p_i, t_j) \in F$ and $M(p_k) = 0 \; \forall \; p_k | (t_j, p_k) \in F$.*

Definition 3 (Firing rule) *An enabled transition may fire. On firing it removes the tokens from all its input places and places one token in each of its output places.*

2.2. Petri net extensions

To make modeling with Petri nets easier and to keep the resulting net sizes small, various extensions have been suggested. The following paragraph lists the most common ones, a detailed discussion can be found in [18] and [25].

- A capacity function $C : P \rightarrow \mathbf{N} \cup \{\infty\}$ determines the number of tokens that is permitted on a certain place. A transition can only fire if, in addition to the enabling rule given above, the output places can take the resulting tokens without exceeding their capacity. The initial marking function is extended to $M_0 : P \rightarrow \mathbf{N}$.

- Weights on the incoming and outgoing arcs of a transition determine the amount of tokens that are removed from a place or written to a place via this arc. In addition, the weight on incoming arcs also prescribes that at least this number of tokens has to be on the respective input place to enable the transition.

- In so-called high-level Petri nets, tokens are distinguishable [11] (e. g. Coloured Petri Nets, (CPN) [16]) by carrying data values of different types. Futhermore, transition guards enable transitions based on token values and token manipulating functions change token values when a transition fires.

- Transitions are annotated with firing durations. Temporal Petri nets can be divided into timed Petri nets, where a time *t* passes between removal of tokens from input places and addition of tokens on output places of a temporal transition, and time Petri nets, where the waiting is done between enabling of a transition and its (atomic) firing, thus allowing disenabling in conflict situations when one token enables several transitions concurrently.

- An inhibitor arc [7] can prevent the transition to which it is connected from being enabled if the place on the origin of the arc contains at least n tokens. In this paper, an inhibitor arc is denoted by a dashed line with a circle at the transition end. While inhibitor arcs greatly facilitate system modeling, they in fact increase the expressiveness of the formalism to such an extend (Turing-equivalence) that formal analysis often becomes impossible. For this reason, Petri net formalisms extended with inhibitor arcs are mostly used in modeling tasks where the resulting net is to be simulated, but not analysed.

Each of the design patterns described later in this paper relies on some of the Petri net extensions presented above. As the modeling projects that are the basis for our involvement with patterns were mostly oriented towards simulation for performance evaluation [15], some patterns make use of inhibitor arcs.

3. Patterns

3.1. History of patterns and previous work

Attempts to capture the building blocks and architectural considerations of a design as so-called "patterns" have their origin in the work of the architect Christopher Alexander [1], and have recently, become more and more popular in the software engineering domain, especially in the context of object-oriented techniques (e. g. [13], [8], [10], [24]).

A pattern in this sense is the description of a recurring problem or problem type and a generalized solution for this problem. Gamma et al. [13] state that a pattern consists of four essential elements: a name, a problem, a solution, and a description of consequences. A suitable and concise pattern name transforms the complex array of elements that make up the solution to an entity in its own right, that can be manipulated and, more importantly, be communicated and discussed as a whole. The problem part of a pattern describes the situation that calls for the application of this particular pattern as well as conditions that need to be met to allow its use. The solution section presents a collection of elements and relationships that are necessary to solve the problem. The designer should regard the "solution" as a starting point and template which he will in most cases need to modify and extend to suit the context of his particular problem set. Finally a pattern may contain a section discussing the consequences of the application of this pattern, the trade-offs and possible alternatives, to allow an informed decision between patterns that solve similar problems. This section will become more and more important after the pattern catalog for a particular design domain has reached a certain size.

Alexander introduced not only patterns, he also suggested to arrange the patterns of a domain in the form of a so-called "pattern language"[1] to systematically guide the designer from the system level specification to a detailed design of a certain quality. This paper however does not yet suggest or discuss a possible Petri net modeling pattern language.

The idea of using patterns, though not known under this name, is in principle not new to the Petri net community. There exists a rather small body of recurring examples to illustrate certain behaviors of a net that can very well be called patterns. However, the problem with those particular examples like deadlock (e. g. [2, 19]), dining philosophers (e. g. [19, 14, 3]), producer/consumer and reader/writer (e. g. [2, 19, 14, 20, 21]) etc. is that, while they do illustrate their particular theoretical point, the style of presentation used is not intended to show how to integrate those examples into models of real systems. The idea of general purpose building blocks is hinted at in [9], but the authors of this case study do not put strong emphasis on the reuse aspect. The presentation of building blocks for communication protocols in [5] and [4] is also heading in the same direction as what we suggest in this paper, but the focus there is more on the demontration of a hierarchy and component concept than on the concise description of fundamental design principles.

3.2. A template for the description of Petri net patterns

One of the most important issues when discussing patterns as design aid is that the pattern description should be detailed, so that there is enough information to apply the pattern, and at the same time it should be possible to quickly survey a number of patterns. Last but not least, pattern descriptions should be uniform to a certain extend to facilitate comparisons between patterns.

We do not claim to know the "correct" way to describe a pattern. Some people may prefer actual application domain examples to explain a pattern, others might like a condensed description of the architectural skeleton. The most suitable form may also depend on the particular pattern itself, its problem context or level of granularity. As a starting point for discussion we suggest the following description template, which follows the four principal elements of a pattern that were mentioned in the previous section. It is, with some modifications, taken from [13]. Certain descriptional sections may not apply to a specific pattern.

Name block:

Name: A name to identify the pattern and distinguish it from others. The name should be such that it clearly conveys the main idea of the pattern.

Problem block:

Problem: The problem and problem context which the pattern addresses.

Example: A concrete example of the problem within some application domain.

Required net formalism : Most of the patterns in this paper rely on the availability of certain extensions of the basic Petri net formalism, such as inhibitor arcs, or CPN-like values on tokens.

Solution block

Solution: The basic idea of the pattern (if non-trivial)

Sample structure: A graphical representation of a Petri net that implements/uses the pattern. The sample structure may be a neutral skeleton or refer to the example given before. In the latter case the description needs to make clear which parts

are fixed elements of the solution and which belong to the example only.

Description: A detailed description of the function of the Petri net building block, also discussing design considerations, variations and options. As far as possible, the explanation should make use of other patterns contained within the pattern under consideration.

Consequences block:

Uses: References to other patterns that are contained within the described pattern.

Similar to: References to and comparisons with other patterns that are similar in some aspect like net structure or targeted problem.

Further sections may be used to highlight special aspects or trade-offs of using a certain pattern.

3.3. Patterns and components

In the following section, sample solutions are presented as complete Petri nets. It might appear as if a logical next step would be to cast those patterns into building-block-like components. While part of the authors' research work is actually concerned with the development of a general and flexible component concept for Petri nets which will also provide features for genericity and parametrization, it must be pointed out that patterns are a more general concept. The structural modifications necessary to adapt a pattern to its actual problem context can generally not be effected by simple component parametrization only.

4. Pattern descriptions

4.1. A-before-B

Name: A-before-B

Problem: On synchronizing two concurrent subnets it may be interesting to know the order in which the enabling tokens arrive at the synchronization point. Of course, this pattern is only useful in temporal Petri nets, so that there exists a notion of "before".

Example: In a manufacturing cell some product is assembled from two components. One component is previously tooled within the same cell and the other is added from another cell with the aid of a robot manipulator. Ideally, the manipulator would deliver component 2 the very moment component 1 gets ready for the assembly, otherwise one component will have to wait for the other. In some realistic situation the manipulator would not wait for component 1 at the end

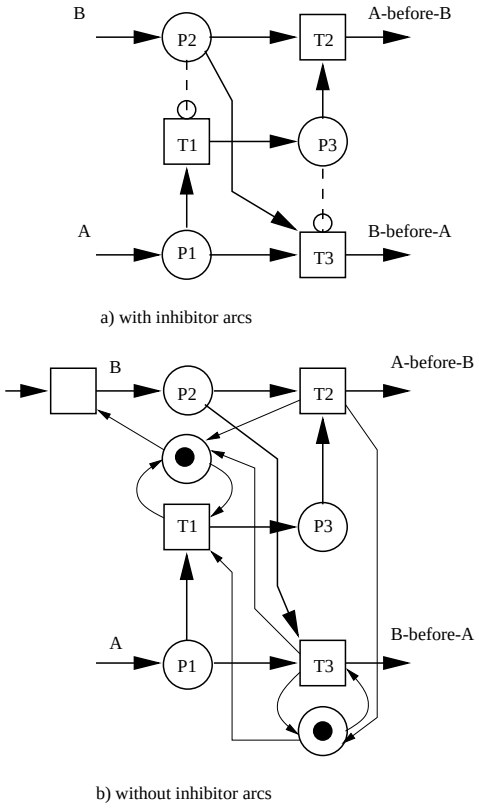

Figure 1. A-before-B

position, but will insert an additional stop on the way. The total delay then does not only depend on the pure time to wait for the slower component to arrive, but also on the the additional time to accelerate the manipulator every time the manipulator had to wait.

Required net formalism: Petri nets with a concept of time.

Solution: The pattern is asymmetrical and consists of two stages: First, it is detected whether token B is already in place when A arrives, and second, there is an additional synchronization so that an early token A has to wait for the later arrival of B (for the other case the arrival of A effects the synchronization).

Sample structure: see Figure 1.

Description: Token A arrives first on `P1`. Only `T1` can fire as `T3` would need a token on `P2` to be enabled. `T1` fires and places one Token on `P3`. Now a token arrives on `P2` and `T2` fires (A before B). Otherwise, token B arrives first on `P2`. Transition `T1` cannot be fired upon arrival of Token A because of the inhibitor arc from `P2` to `T1`, therefore `T3` fires (B before A). The additional inhibitor arc between `P3` and `T3` is necessary to

prohibit the firing of T3 before T2 in case of an arrival sequence A-B-A.
If this is not already enforced by the surrounding net, it will be necessary to restrict the capacities of places P1 and P2 to $C(P1) = 1$ and $C(P2) = 1$. Figure 1(b) shows how the pattern can be implemented using a Petri net formalism without inhibitor arcs. For high-level Petri nets with distiguishable tokens this pattern may be extended to deliver always the first (or the second) token at the output.

4.2. Token multiplier

Name: Token multiplier

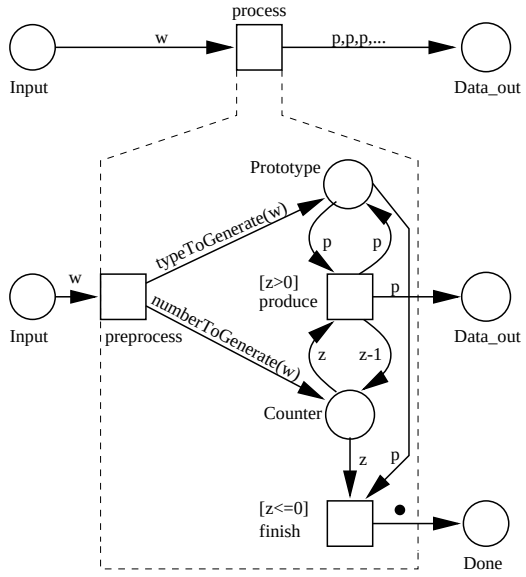

Figure 2. Token multiplier

Problem: With the firing of a certain transition it is desired to produce several tokens on the outgoing arc. The number of tokens to be produced is determined by an input token value at run time, so constant arc weights cannot be used to achieve this purpose.

Example: For the model of a piece of semiconductor manufacturing machinery it is necessary to generate the tokens representing the dies from one token symbolizing the wafer. For this model there can be a variety of wafer tokens that differ in the value of the wafer diameter attribute. From the wafer diameter a function in the transition calculates the corresponding number of dies, this number is therefore only known at run time.

Required net formalism: Coloured Petri nets or comparable high-level Petri net formalisms (necessary are value tokens, guard conditions and arc expression functions)

Solution: The basic idea is to model desired behavior of the single transition (process in Fig. 2) using a loop-like structure where a token, the value of which is decremented in each iteration, acts as loop counter.

Sample structure: see Fig. 2

Description: The pattern requires three items of input information: a value from which the *number* of tokens that should be generated can be derived, a *prototype* of the tokens to be produced, and a *starting signal*. In the example structure of Fig. 2, the input token acts as starting signal and also provides the other items of information. The preprocess transition calculates the number of tokens that will be generated, extracts the prototype information from the input, and distributes the the resulting tokens to the Counter and Prototype places respectively. Each subsequent firing of the produce transition places a copy of the prototype into the Data_out place and decrements the value of the loop counter token stored in Counter. When the loop counter value reaches zero, the transition finish is enabled instead of produce. Its firing clears the Counter and Prototype places. Additionally, a completion signal can be generated.

For the example stated above, w would contain information about the wafer and the prototype would correspond to the token representation of a die.

Variations to the pattern may deal with different ways to provide the input information, to generate the output tokens from the prototype, or to include additional processing steps for each iteration.

Similar to: This pattern is similar to the Petri net extension of "marking controlled arc weights" suggested in [3],[22] and [12]. Whereas these refer to a notational extension of the basic Petri net formalism that determines the weight of an arc by the number of tokens in a certain place, we discuss here a more flexible, architectural approach to the solution of the underlying problem.

Uses: The structure of the net in Fig. 2 can be regarded as composed from even simpler patterns: Prototype and produce form a *producer loop*, Counter and produce form a *counter loop*, and the combination of Counter with the alternatively enabled transitions produce and finish is a *deterministic branch* in the control flow.

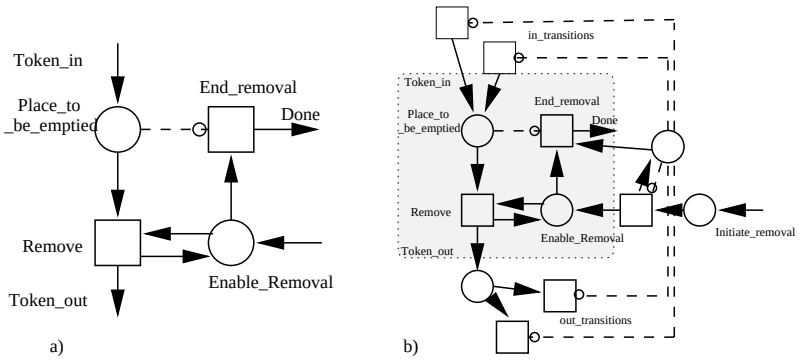

Figure 3. Complete removal a) basic pattern b) without priority support in the formalism

4.3. Complete removal

Name: Complete removal

Problem: All tokens stored in a particular place are to be removed from that place while the number of tokens is not known a priori.

Example: see pattern "quasi-continuous movement" later in this paper

Required net formalism: Petri nets with inhibitor arcs [7]. If the intention is to remove all tokens en bloc from the place before the tokens that were removed first cause any other transition firings in the net then it is either necessary to utilize some transition priority concept provided by the formalism or the pattern needs to be embedded in an environment like the one shown in Figure 3 b). All inhibitors except for the on between `Place_to_be_emptied` and `End_removal` can be substituted with auxiliary constructions like in the A-before-B pattern.

Solution: A transition is fired repeatedly and each time removes one token from the place to be emptied. This continues until the place is empty.

Sample structure: see Figure 3

Description: After a token is placed into `Enable_removal` the `Remove` transition can fire. Because the `Enable` token is returned to the `Enable removal` place after each firing, the transition can fire repeatedly, until there are no more tokens in `Place to be emptied`. Then the inhibitor arc enables the `End removal` transition which removes the token from the `Enable removal` place.

Similar to: This pattern is similar to the concept of "reset arcs" ("Abräumkanten") [3] or "erase arcs" [7]. The "complete removal" pattern can be regarded as a replacement of reset arcs valid for Petri nets extended only with inhibitor arcs. "Erase arcs" are introduced mainly to realize some sort of reset functionality for parts of a system, consequently erasing is regarded as an atomic operation, and the tokens taken from the "erase place" are destroyed. The "complete removal" pattern on the other hand is more flexible as it allows to embed additional actions into the removal process and the tokens removed from the place are not necessarily lost.

4.4. Quasi-continuous movement

Name: Quasi-continuous movement

Problem: A token progressing through several processing stages is usually modeled using a simple sequence of places and transitions. This is not possible if either the number of discrete positions varies at run-time, the processing stages are partly overlapping, the speed of progress is variable or a quasi-continuous movement from one processing station to the next has to be modeled. More generally, this pattern may be used to model all sorts of problems where numerical attributes of a collection of tokens are changing synchronously, transitions are to be fired whenever those attributes reach certain values, and where the amount of change per step or the limit values to fire the transitions are only known at run-time.

Example: We consider the model of an assembly line (see Figure 5) with a conveyor belt of fixed physical length and one or more processing stations on which a (continuous) stream of parts to be processed is flowing, one immediately after the other. The parts moving on this assembly line (represented as tokens) are assumed to be of variable length so that the position of a certain

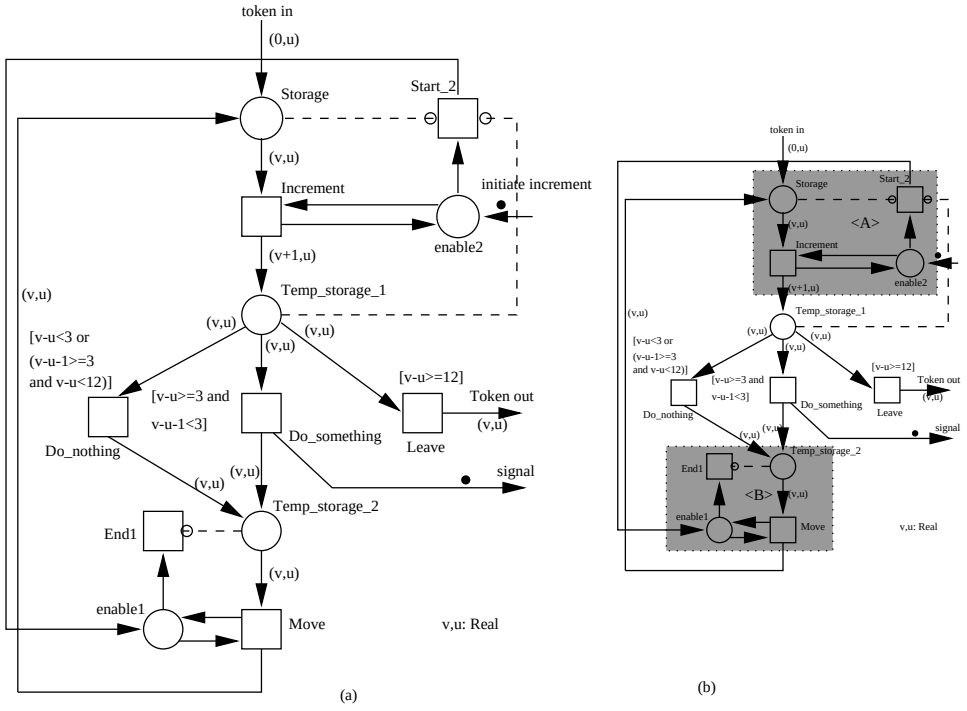

Figure 4. a) Quasi-continuous movement b) Pattern "complete removal" within pattern "quasi-continuous movement"

part can not be given in terms of it being the n-th part counting from the start of the line, but only in terms of the numerical position of e. g. the front edge of the part. Each processing station then has a certain physical range on the belt within which it can operate on parts.

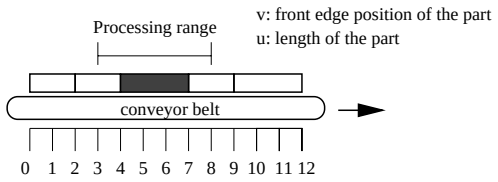

Figure 5. Conveyor belt with transported parts of varying size

Required net formalism: Coloured Petri nets (numerical token attributes), inhibitor arcs

Solution: The "quasi-continuous movement" pattern does not use a sequence of places and transitions to model the position of the parts on the assembly line but circulates the tokens in a loop with one transition to increment the token attribute value and other guarded transitions to initiate actions for tokens whose attribute values fall within a range specified in the guard predicate.

Sample structure: see Figure 4

Description: The pattern consists mainly of two applications of the "complete removal" pattern and a deterministic branching stage. In the first application of "complete removal" (places '`Storage` and `Enable_2` with transitions `Increment` and `Start_2`) (Figure 4 (b), "A") there is an additional inhibitor arc necessary between `Temp_storage_1` and `Enable_2` to ensure that all tokens have passed the branching stage before they get pumped back to the `Storage` place. The second application (Figure 4 (b), "A") (places `Temp_storage_2` and `Enable_1` with transitions `Move` and `Start_1`) uses the unmodified basic pattern.

Uses pattern: Complete removal

5. Conclusion

In this paper a style of presentation for design knowledge in Petri net systems modeling was suggested, which

is by no means complete or finished - on the contrary, we hope to initiate a discussion among Petri net users about how to present, categorize, and select this kind of knowledge for modeling tasks in the various fields where Petri nets might be used.

There is still much work to be done before a complete catalog of Petri net design patterns is available to be applied to practical modeling tasks. We see the most important areas as follows:

- Definition of appropriate, domain specific pattern presentation styles and templates.
- Collection, identification, abstraction, and categorization of the existing body of Petri net design knowledge and the patterns behind it.
- Design of a sufficiently general Petri net language allowing for powerful mechanisms for composition and parametrization. The experience from and with different Petri net languages [6], [11], [16], [17], [23] must be taken into account as well as the huge body of knowledge from programming language design.
- Design of a pattern language, at least for a restricted set of patterns.

Experience shows that application knowledge of this kind enhances the acceptance of a tool or a language. Not only does it raise the productivity of individuals using it, a systematic categorization of design experience also promotes a common terminology and thus makes communication and discussion much more efficient. This paper is intended as a contribution towards this end.

References

[1] C. Alexander, S. Ishikawa, M. Silverstein, M. Jacobson, I. Fiksdahl-King, and S. Angel. *A Pattern Language*. Oxford University Press, 1977.

[2] H. Balzert. *Lehrbuch der Software-Technik*, volume 1. Spektrum Akademischer Verlag, 1996.

[3] B. Baumgarten. *Petri-Netze, Grundlagen und Anwendungen*. Spektrum Akademischer Verlag, 1996.

[4] B. Baumgarten, H. J. Burkhardt, P. Ochsenschläger, and R. Prinoth. The signing of a contract - a tree-structured application modelled with petri net building blocks. In G. Goos and J. Hartmanis, editors, *Advances in Petri Nets 1985*, number 222 in Lecture Notes in Computer Science. Springer, 1985.

[5] B. Baumgarten, P. Ochsenschläger, and R. Prinoth. *Protocol Specification, Testing, and Verification*, chapter Building Blocks for Distributed System Design, pages 19–38. Elsevier Science Publishers B.V. (North-Holland), 1986.

[6] D. Buchs and N. Guelfi. CO-OPN: A concurrent object-oriented Petri net approach. In *Proceedings of the 12th International Conference on the Application and Theory of Petri Nets*, 1991.

[7] H. J. Burkhardt, P. Ochsenschläger, and R. Prinoth. Product nets, a formal description technique for cooperating systems. GMD-Studien 165, GMD, Gesellschaft für Mathematik und Datenverarbeitung mbH, 1989.

[8] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. *Pattern-Oriented Software Architecture - A System of Patterns*. Wiley and Sons, 1996.

[9] A. Caloini, G. A. Magnani, and M. Pezze. Software design of robot controllers with petri nets: a case-study. In *Proceedings of the 1996 IEEE International Conference on Systems, Man and Cybernetics*, 1996.

[10] J. O. Coplien and D. C. Schmidt, editors. *Pattern Languages of Program Design*. Addison-Wesley, 1995.

[11] R. Esser. *An Object Oriented Petri Net Approach to Embedded System Design*. PhD thesis, ETH Zurich, 1996.

[12] H. Fuss. P-T-Netze zur numerischen Simulation von asynchronen Flüssen. In J. H. G. Goos, editor, *GI - 4. Jahrestagung, Berlin Oktober 1974*, number 26 in Lecture Notes in Computer Science, pages 326–335. GI, Gesellschaft für Informatik, Springer-Verlag, 1975.

[13] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns, Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[14] R. G. Herrtwich and G. Hommel. *Nebenläufige Programme*. Springer Verlag, 1994.

[15] J. W. Janneck and M. Naedele. Modeling a die bonder with petri nets: A case study. *IEEE Transactions on Semiconductor Manufacturing*, Aug. 1998.

[16] K. Jensen. *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use*, volume 1: Basic Concepts, of *EATCS Monographs in Computer Science*. Springer-Verlag, 1992.

[17] C. A. Lakos. Object Petri nets - definition and relationship to coloured nets. Technical Report TR94-3, Computer Science Department, University of Tasmania, 1994.

[18] T. Murata. Petri nets: Properties, analysis, and applications. *Proceedings of the IEEE*, 77(4):541–580, Apr. 1989.

[19] J. L. Peterson. *Petri Net Theory and the Modeling of Systems*. Prentice Hall, 1981.

[20] W. Reisig. *Petri Nets, An Introduction*. Springer-Verlag, 1985.

[21] W. Reisig. *A Primer in Petri Net Design*. Springer-Verlag, 1992.

[22] R. Valk. Self-modifying nets, a nautural extension of petri nets. In J. H. G. Goos, editor, *Automata, Languages and Programming, Fifth Colloquium, Undine, July 17-21, 1978*, number 62 in Lecture Notes in Computer Science, pages 464–476. Springer-Verlag, 1978.

[23] R. Valk. On processes of object Petri nets. Technical Report 185, Fachbereich Informatik, Universität Hamburg, 1996.

[24] J. M. Vlissides, J. O. Coplien, and N. L. Kerth, editors. *Pattern Languages of Program Design 2*. Addison-Wesley, 1996.

[25] R. Zurawski and M.-C. Zhou. Petri nets and industrial applications: A tutorial. *IEEE Transactions on Industrial Electronics*, 41(6):567–583, Dec. 1994.