REUSING DESIGN EXPERIENCE FOR PETRI NETS THROUGH PATTERNS

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

KEYWORDS: *Petri nets, design patterns, modeling*

ABSTRACT

Industrial applications of Petri nets for modeling and design often result in very complex models (Zurawski and Zhou 1994; Esser, Janneck, and Naedele 1997). Designers can handle this complexity much better if they can (re)use structures expressing expert modeling experience at a higher level of design and abstraction than the basic elements. In the software engineering domain this observation led to the recent trend of using software design patterns (Buschmann and Meunier 1995; Gamma et al. 1995). Our experience with Petri nets, an established and well researched visual language for systems modeling, simulation, and analysis (Murata 1989; Reisig 1992; Zurawski and Zhou 1994), in projects concerned with modeling of manufacturing machines (Esser, Janneck, and Naedele 1997; Naedele and Janneck 1998) and integrated circuits (Gries 1998), shows that patterns are very useful in this area as well. In this paper we continue work presented in (Naedele and Janneck 1998) describing a template to capture, document, and present design knowledge in the form of design patterns. Finally, the example of the modeling of an integrated circuit shows an application of that template.

1 INTRODUCTION

Petri nets (Murata 1989; Reisig 1992), (Zurawski and Zhou 1994) are an established visual formalism that can be applied in various domains such as economics, mechanics, work flow organisation, theoretical computer science, and hardware as well as software design of complex concurrent computer systems. Petri nets are chosen for these tasks since they are able to model a variety of processes as data, control, event, and material flow in heterogeneous systems.

Industrial applications of Petri nets (Zurawski and Zhou 1994) often result in very complex models (e.g. see the screen shot of the CodeSign (Esser 1996) Petri net modeling and simulation tool in Fig. 1 from (Gries 1998)). Such complex models are difficult to create and to understand. Both tasks are facilitated if creation and understanding of the model are not attempted in terms of basic elements (places, transitions, and arcs), but in terms of more coarse-grained structures.

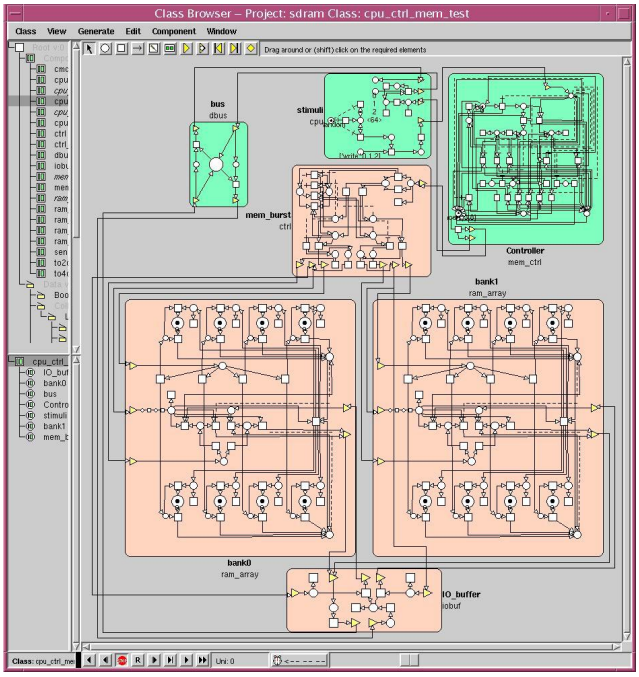

Figure 1: Model of a PC memory system.

In the software engineering domain this observation led to the recent trend of using software de-

sign patterns (Buschmann et al. 1996; Gamma et al. 1995). Our experience with Petri nets in projects concerned with the modeling of manufacturing machines (Esser, Janneck, and Naedele 1997) and integrated circuits (Gries 1998) (Fig. 1) suggests that patterns are very useful in this area as well. Since complex, heterogeneous systems combine many different kinds of processes simultaneously, patterns should be used for the modeling of such systems to capture, document, and transfer domain knowledge between people from different fields of expertise. In (Naedele and Janneck 1998) we presented an approach to capture such design knowledge following the traditions of the design pattern community.

The purpose of this paper is to demonstrate how a complex model of a computing system can be structured and thus made more easily understandable through the use of patterns as proposed in (Naedele and Janneck 1998). By using patterns, which combine the knowledge of experts from different domains about modeling styles and methods, systems can be modeled easily by other people. As an example, section 3 will describe some of the patterns used in the Petri net model of a memory system from (Gries 1998) shown in Fig. 1.

In order to be able to reference and apply patterns, we will use a template for Petri net patterns which was introduced with some more complex patterns in (Janneck and Naedele 1998) and which is based on the template suggested in (Gamma et al. 1995). An explanation of this template is given in section 2.

2 PATTERNS

2.1 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 (Alexander et al. 1977) and have recently become more and more popular in the software engineering domain, especially in the context of object-oriented techniques (e.g. (Buschmann et al. 1996; Gamma et al. 1995)).

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. (Gamma et al. 1995) state that a pattern consists of four essential elements:

- a suitable and concise pattern name,
- a problem statement, describing the situation and boundary conditions for using a particular pattern,
- the solution section, presenting a structure of elements and relationships to solve the problem,
- 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.

The idea of using patterns, though not under this name, is in principle not new to the Petri net community. There exists a 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. (Peterson 1981)), dining philosophers (e.g. (Peterson 1981; Baumgarten 1996)), producer/consumer, and reader/writer (e.g. (Peterson 1981; Reisig 1992)) etc. is that, while they do illustrate their particular theoretical point, the style of presentation used is not intended to show how to use those examples in the models of complex systems. The idea of general purpose building blocks is hinted at in (Caloini, Magnani, and Pezze 1996), 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 (Baumgarten, Ochsenschläger, and Prinoth 1986) and (Baumgarten et al. 1985) is also heading in the same direction as our suggestion in this paper, but the focus there is more on the demonstration of a hierarchy and component concept than on the concise description of fundamental design principles.

2.2 A template for the description of Petri net patterns

The most suitable form for a template to describe patterns will depend on the particular pattern itself, its problem context or level of granularity, and of course the needs of the respective pattern users. 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 (Gamma et al. 1995). 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 formalism: The minimal/most simple modeling Petri net variant that is required to realize the pattern. This information is particularly important if a formalism allows special features.

Solution block:

Solution: The basic idea of the pattern.

Sample structure: A graphical representation of a graphical structure 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 pattern elements, 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, variations, or trade-offs of using a certain pattern.

3 PATTERN DESCRIPTIONS

In this section we will illustrate the pattern concept by describing a few simple but useful recurring structures when modeling systems with Petri nets. We will first introduce low-level patterns (which would be termed *idioms* in (Buschmann et al. 1996)) and then present a pattern on a somewhat larger scale. For some even more complex patterns, see for example (Janneck and Naedele 1998).

The first pattern is a useful idiom for expressing choices that would be expressed by conditional constructions in programming languages.

Name: Deterministic choice

Problem: A token is to be processed by exactly one of a given number of processing subnets. The *value* of the token determines the branch that is chosen.

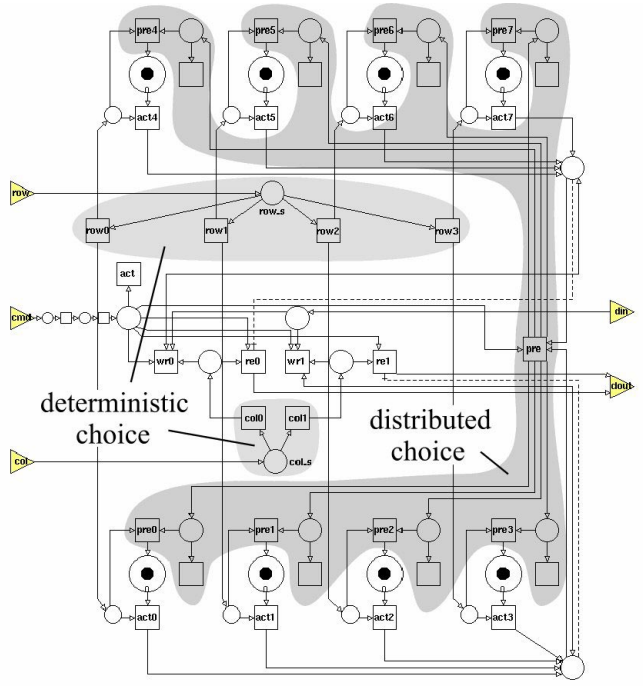

Figure 2: A model of a memory bank.

Example: In a random access memory (RAM), the memory cell activated depends on the *address* given to the memory unit. Once the address is available, one of several subnets representing the respective portion of the RAM has to be activated for further processing.

Required formalism: High-level Petri nets.

Solution: The token will be processed by transitions which are in structural conflict. Appropriate guards will resolve this conflict in a deterministic way.

Sample structure: See the decoding of the column and row addresses at the places *col_s* and *row_s* by the associated transitions in Fig. 2.

Description: In Fig. 3, the token to be processed would reside in the place. The processing subnets are connected to that place by transitions t_i that are guarded by predicates G_i in such a way that for any possible value of the token exactly one G_i will be true, i.e. exactly one transition t_i will be enabled. Often, the requirement that *exactly* one guard must be true will imply the addition of a default transition.

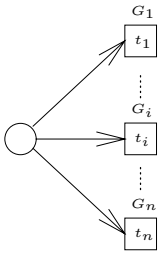

Figure 3: Deterministic choice pattern.

Variation: Relaxing the above requirement by allowing more than one guard to be true obviously leads to a non-deterministic choice between transitions enabled by the same token.

Trade-offs: The structural conflict between the guarding transitions assumes some kind of (implicit or explicit) synchronization of their access to the input place. If this mechanism is not to be abstracted from, it might be more appropriate to use the *distributed choice* pattern.

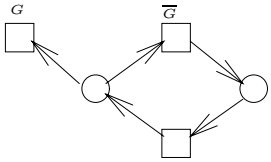

Figure 4: A terminated loop.

The next pattern applies deterministic choice to the problem of terminating an iteration.

Name: Terminated loop

Problem: Iterate a certain sequence of actions until a condition becomes true.

Example: When a RAM is ready to accept data, the CPU must transfer a certain amount of data according to a fixed timing scheme within a certain amount of clock cycles.

Required formalism: High-level Petri nets.

Solution: A circular net structure with a deterministic choice terminating the circle.

Sample structure: See the transitions *continue*, *terminate*, and *delay* of the CPU model in Fig. 5 which control the writing of data to the RAM.

Description: The loop is implemented by a circular net, like in the simple case in Fig. 4. There is a deterministic choice between the termination transition (guarded by G in the figure) and the continuation transition (guarded by $\overline{G}$). Once G becomes true, the token is removed from the loop and processing terminates.

Uses: Deterministic choice

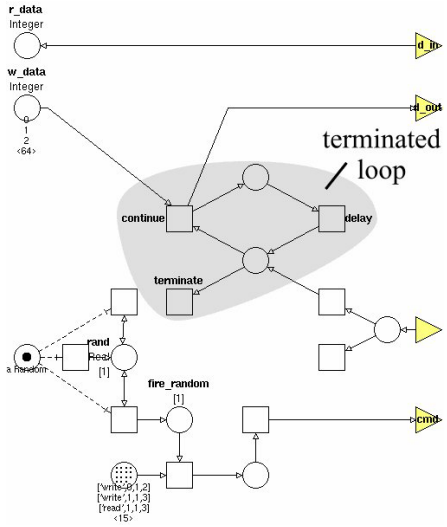

Figure 5: A terminated loop in context.

Finally, the last pattern shows how to implement a behavior similar to the deterministic choice in a situation where branches that may be chosen are distributed and where direct structural conflict is not an appropriate modeling technique.

Name: Distributed Choice

Problem: In a distributed situation, the basic deterministic choice pattern is not applicable, since distributed components cannot be in direct conflict on a given place.

Example: In the shaded area in Fig. 2, RAM cells are physically distributed over a bank, and their selection basically constitutes a deterministic choice. It seems inappropriate, however, to model the selection transitions as being in direct conflict to each other.

Required formalism: High-level Petri nets.

Solution: After broadcasting the token to all possibly affected units, each unit deterministically determines whether it has to process the token.

Sample structure: The broadcast and selection structure in the shaded area in Fig. 2.

Description: In Fig. 6, incoming tokens are buffered in each subnet and are only forwarded to the rest of the subnet if G_i is true. They are removed from the buffer if $\overline{G_i}$ is true. As in the deterministic choice pattern, we assume only (or exactly) one of the guards G_i to be true for any given token.

Trade-offs: While this pattern is functionally similar to the *deterministic choice* pattern, it is applicable to distributed situations, i.e. the guarding transitions may not be assumed for some reason to have direct access to the input place. This is the case, e.g., when there is no direct, atomic, and synchronized access of the processing paths to a single memory location, as in a networked environment. Note also that relaxing the requirement that only one G_i be true for any given token does *not* result in non-deterministic choice but rather in concurrent execution of more than one of the branches. Furthermore, no special default branch has to be introduced in order to prevent the accumulation of unconsumed tokens in the input place.

Uses: Deterministic choice.

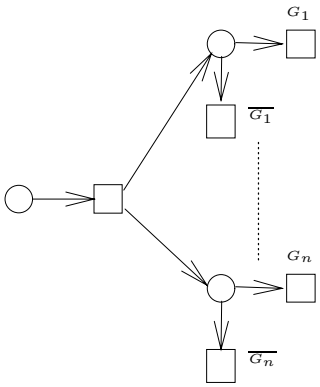

Figure 6: Distributed choice pattern.

4 RESULTS

Finally, we will give some results concerning the usage of patterns in the memory system model presented in (Gries 1998). The model can be subdivided into three main areas: memory components (RAM and buffer), which use data and control flow, controllers (external memory controller and controller on the memory chip), which can be best described by control flow, and stimuli parts (CPU and bus components) which are also control flow dominated parts.

system blocks	overall used		covered by patterns	
	places	trans.	places	transitions
RAM + buffer	73	83	28	60
Controllers	25	42	11	29
CPU + bus	13	12	3	5
$\sum$	111	137	42 (37.8%)	94 (68.6%)

Table 1: Places and transitions of the memory system model covered by the three introduced patterns.

Table 1 summarizes how many places and transitions of the model are already represented by the application of the three mentioned patterns only, i.e., 37.8% of all places and 68.6% of all transitions are covered by them.

Table 2 shows the usage of the patterns in the components of the model. In particular, the control flow components require the deterministic choice pattern for the representation of basic address and instruction decoding processes. Besides, terminated loop patterns can be found in the controllers since they are responsible for the exact timing of read, write, and refresh actions of the RAM. Finally, distributed choice patterns are used to model the activation of distributed components within a larger one, e.g. RAM cells within a RAM or separate address decoders within a controller.

system blocks	term. loop	distr. choice	det. choice
RAM + buffer	0	2	8
Controllers	4	2	5
CPU + bus	1	0	1

Table 2: Patterns used by the memory system.

5 CONCLUSION

In this paper we show how patterns can be used to structure complex models, which are thus easier to create and understand. Just like in the area of software engineering and architecture, we believe that design-

ing and documenting systems using patterns fosters reuse, efficiency, robustness, and understandability of even very complex Petri net models. This would clearly enhance the applicability of the Petri net approach to complex modeling tasks.

This paper is intended to be a starting point for further discussion among users of Petri nets and other flow-oriented visual languages about how to select, categorize, and present existing knowledge for modeling tasks.

References

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

Baumgarten, B. (1996). *Petri-Netze, Grundlagen und Anwendungen*. Spektrum Akademischer Verlag.

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

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

Buschmann, F. and R. Meunier (1995). A system of patterns. In *Pattern Languages of Program Design*.

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

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

Esser, R. (1996). *An Object Oriented Petri Net Approach to Embedded System Design*. Ph. D. thesis, ETH Zurich.

Esser, R., J. W. Janneck, and M. Naedele (1997). Applying an object-oriented petri net language to heterogeneous systems design. In *Petri Nets in Systems Engineering*.

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

Gries, M. (1998, June). Modeling a memory subsystem with petri nets: a case study. In *Workshop Hardware Design and Petri Nets HWPN98*, Lisbon, Portugal, pp. 186–201.

Janneck, J. W. and M. Naedele (1998). Introducing design patterns for petri nets. Technical report, Computer Engineering and Networks Lab (TIK), Swiss Federal Institute of Technology Zurich (ETH Z).

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

Naedele, M. and J. W. Janneck (1998). Design patterns in Petri net system modeling. In *Proceedings ICECCS'98*, pp. 47–54.

Peterson, J. L. (1981). *Petri Net Theory and the Modeling of Systems*. Prentice Hall.

Reisig, W. (1992). *A Primer in Petri Net Design*. Springer-Verlag.

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

BIOGRAPHIES

Matthias Gries studied electrical engineering and specialised in digital signal processing. He received his MS degree from the Technical University of Hamburg-Harburg, Germany, in 1996. Currently, he is working at the Swiss Federal Institute of Technology (ETH), Zürich, Switzerland, as a research assistant in the Computer Engineering and Networks Lab (TIK).

His research project covers the design of an ATM switch where he is responsible for the design of a scalable, high speed shared memory architecture using recent memory technologies.

Jörn W. Janneck graduated in computer science from the University of Bremen, Germany, in 1995, working on computer graphics and artificial intelligence.

From 1995 to 1996 he worked as a researcher for the Fraunhofer Institute for Material Flow and Logistics (IML) in Dortmund, Germany, in the field in business modeling and simulation. Since 1996 he is holding a position as a research assistant at the Swiss Federal Institute of Technology (ETH), Zurich, Switzerland, doing research on visual language semantics and various aspects of discrete event systems including simulation, formal analysis, and modeling languages such as Petri nets.

Martin Naedele studied electrical engineering at Ruhr-University, Bochum, Germany, and Purdue University, West Lafayette, IN. He received his MS degree in electrical engineering from Ruhr-University in 1997. Currently, he is working at the Swiss Federal Institute of Technology (ETH), Zürich, Switzerland, as PhD student and research assistant in the Computer Engineering and Networks Lab (TIK).

His research interests include embedded systems, fault-tolerant real-time computing, software engineering, and computer security.