Association for Information Systems

AIS Electronic Library (AISeL)

Wirtschaftsinformatik 2024 Proceedings Wirtschaftsinformatik

2024

Addressing the Schema Representation Problem in Process Models Using Petri Nets – First Results Illustrated by the Dining Philosophers Problem

Sebastian Stephan
German Research Center for Artificial Intelligence (DFKI); Saarland University, sebastian.stephan@dfki.de

Josip Lovrekovic
German Research Center for Artificial Intelligence (DFKI); Saarland University, josip.jelicic-lovrekovic@dfki.de

Peter Fettke
German Research Center for Artificial Intelligence (DFKI); Saarland University, peter.fettke@dfki.de

Follow this and additional works at: https://aisel.aisnet.org/wi2024

Recommended Citation
Stephan, Sebastian; Lovrekovic, Josip; and Fettke, Peter, "Addressing the Schema Representation Problem in Process Models Using Petri Nets – First Results Illustrated by the Dining Philosophers Problem" (2024). *Wirtschaftsinformatik 2024 Proceedings*. 105.
https://aisel.aisnet.org/wi2024/105

This material is brought to you by the Wirtschaftsinformatik at AIS Electronic Library (AISeL). It has been accepted for inclusion in Wirtschaftsinformatik 2024 Proceedings by an authorized administrator of AIS Electronic Library (AISeL). For more information, please contact elibrary@aisnet.org.

Addressing the Schema Representation Problem in Process Models Using Petri Nets – First Results Illustrated by the Dining Philosophers Problem

Research in Progress

Sebastian Stephan, Josip Lovrekovic, and Peter Fettke

German Research Center for Artificial Intelligence (DFKI) and Saarland University, Saarbrücken, Germany
{sebastian.stephan,josip.jelicic-lovrekovic,peter.fettke}@dfki.de

Abstract. In this paper, we introduce the schema representation problem using the example of dining philosophers: The difference between a system model of five eating philosophers and a schema model for a set of eating philosophers is of major importance. In a Petri net model, each philosopher and each fork would be considered as separate entities with their relating states and transitions. However, this approach lacks due to scalability and dynamic behavior, as adding more philosophers and forks significantly increases the model's size. To model any set of dining philosophers, a Petri net schema is useful. However, there is no modeling technique to model an infinite set of philosophers and forks, and to access its single elements. To address this problem, we provide the *elm*-notation, which allows us to dynamically unfold and aggregate any sets whereby behavior can be described for each philosopher and fork on schema level.

Keywords: Petri nets, system model, schema, behavioral modeling, *elm*-notation

1 Introduction

In modeling, it is of major interest to distinguish between a *system model* and a *schema* for system models (Fettke & Reisig 2022*a*). Typically, a modeler begins shaping system behavior based on concrete runs. Once detailed, the modeler adopts a universal interpretation, transforming runs into a complete system model, ensuring the system only behaves as described by its structure (Desel 2008, Fahland 2009).

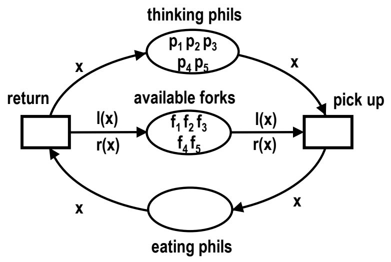
A system model is intended to represent just one concrete system with its finite sets of real or imagined world items and data objects. Using a so-called *term language* as presented in Genrich & Lautenbach (1981) or Reisig (1991), the structure can be abstracted from the system model. This makes it possible to describe a set of similar system models represented by a *schema*. Analogous to an algebra and its interpretation, by changing the interpretation, the net behavior can be changed without becoming externally visible (Starke 1990). Usually, a structure comes with a signature, providing a symbol for each of the structure's components like domains, functions, predicates, constants, and propositions (Fettke & Reisig 2022*a*,*b*). Based on system models that describe a concrete domain, the question now arises how the behavior of each element can be taken into account in a schema model. A symbolic representation of the set itself would be inaccurate and also incorrect, as this would represent the set itself as a token and system behavior is not described by its single elements. This does not allow a dynamic handling of sets where elements are produced, event-driven transformed, and then synchronized in a structured manner (Fettke & Reisig 2024).

Thus, the problem we are focusing on is concerned with finding a suitable technique dealing with set symbols in a Petri net schema (schema representation) that accurately and efficiently depicts both structures and behavior of similar system models, but at the same time remains flexible, understandable and manageable for design and analysis. Before we present the solution more in detail, we motivate the schema representation problem using the example of the dining philosophers (Dijkstra 1971) in section 2. In section 3, we introduce the *elm*-notation provided by the modeling infrastructure HERAKLIT (Fettke & Reisig 2024) and describe how it contributes to solve the motivated problem. Section 4 gives an overview of related work. Finally, section 5 summarizes the key findings and provides an outlook on our further research work.

2 Motivating Example

The dining philosophers problem illustrates common pitfalls and challenges associated with designing concurrent systems (Dijkstra 1971). It describes a situation in which five philosophers are thinking and eating. Sitting around a table with five forks, each philosopher needs his left and right fork to eat. This illustrates the challenges of synchronization and deadlocks in distributed systems. A run of the system would involve setting specific numbers of places and transitions for each philosopher and fork (Reisig 2007). While the state and behavior of each element is modeled explicitly, a system model makes the problem more descriptive, but also complex and large for extensive systems, which may include hundreds of philosophers and forks. Another possibility how to describe the dining philosophers is to use a structure, which assembles real world items, data, constants, functions, predicates and propositions that can be updated, generated, deleted, computed and transformed by a system (Fettke & Reisig 2022*a*).

As depicted in Figure 1, a structure with five philosophers and forks is given. As shown, the system model is limited with regard to its expressiveness. First, only five philosophers and forks are considered. While this type of modeling works for domains with a finite set of elements, this approach fails for domains that are often not entirely known during modeling. Second, the concrete amount of elements has to be modeled in the places *thinking phils* and *available forks*. However, this makes Petri nets less readable and complex, especially when dealing with similar structures which might contain hundreds of different philosophers. Due to the close coupling of the structure to the system model, as many system models as there are structures would be needed. Third, given the two functions *l(x)* and *r(x)*, the case that only one philosopher (x) can eat with two forks, namely with his left and right fork, is reflected. This means that the dependency of which philosopher is allowed to eat with which fork is explicitly described in the model. Changes to the system behavior, for example to allow a philosopher to eat with any subset of forks (e.g. eating with only one or even three forks), need to be changed directly in each system model and structure respectively.



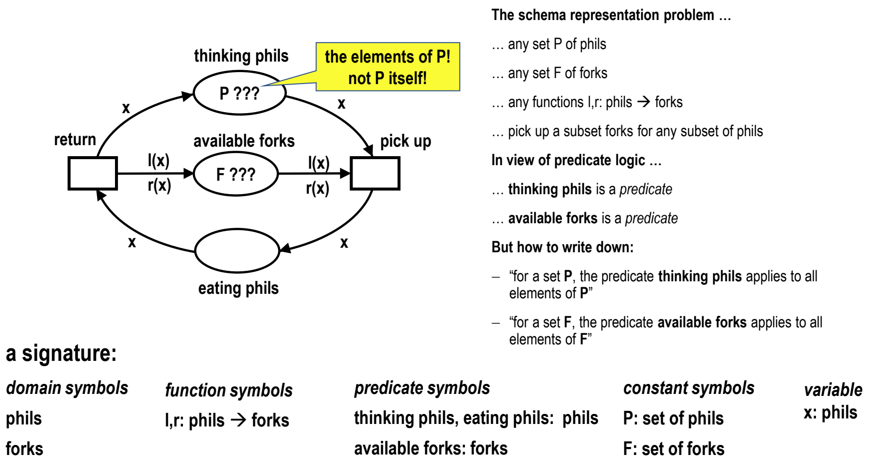
a structure:

domains	*functions*	*predicates*	*constants*	*variable*
phils = $\{p_1, p_2, p_3, p_4, p_5\}$	l,r: phils → forks	thinking phils, eating phils: phils	p_1, p_2, p_3, p_4, p_5: phils	x: phils
forks = $\{f_1, f_2, f_3, f_4, f_5\}$	$l(p_i) = f_i$	available forks: forks	f_1, f_2, f_3, f_4, f_5: forks	
	$r(p_i) = f_{i+1}$			
	$f_6 =_{def} f_1$			

Figure 1. System model of the dining philosophers (based on Reisig (2023*b*))

In order to represent similar structures by a model, the question now arises how any (in)finite set of philosophers and forks can be modeled in a Petri net model, for which any subset of philosophers can pick up any subset of forks. One possibility could be to further abstract the system behavior using a *schema*. Usually, a structure comes with a signature, which provides a symbol for each of the structure's components (Fettke & Reisig 2022*a*,*b*). For the dining philosophers, each domain gets its symbol (*phils* and *forks*) denoted as its type, analogous to the types of components of structures. As depicted in Figure 2, the schema model consists of two domain symbols, one function symbol, three predicate symbols, and two constant symbols. By the two domain symbols, an arbitrary set of philosophers and forks can be modeled and represented symbolically in the places. This kind of modeling means that the symbols *P* and *F* hold the complete set of philosophers and forks, respectively. However, representing a set of philosophers and forks in one token each does not correspond to the system behavior we want to describe. Behavior resulting from the single elements of a set is not reflected properly. This is also reflected in the functions *l(x)* and *r(x)*, which assign a left and right fork for a given philosopher. In the case that every philosopher can only eat with a left and right fork, this type of modeling might be sufficient and more descriptive. However, a more general representation that makes it possible to return the corresponding number of forks depending on the philosopher is not possible. For example, there might be a system model where philosophers can eat with only one fork, or even more. In other words, behavior like picking up or returning any subset of forks for any subset of philosophers is not described in such a schema model.

To the best of our knowledge, there is no suitable technique to address this modeling issue, which we refer to as the *schema representation problem*. In general, the universal quantifier of predicate logic is missing on modeling level not only to describe rules or conditions that apply to all elements of a given set symbol, but also to access them individually and describe their behavior, and thus consider additional concurrency.



The schema representation problem …

… any set P of phils

… any set F of forks

… any functions l,r: phils → forks

… pick up a subset forks for any subset of phils

In view of predicate logic …

… **thinking phils** is a *predicate*

… **available forks** is a *predicate*

But how to write down:

– "for a set **P**, the predicate **thinking phils** applies to all elements of **P**"

– "for a set **F**, the predicate **available forks** applies to all elements of **F**"

a signature:

domain symbols	***function symbols***	***predicate symbols***	***constant symbols***	***variable***
phils	**l,r: phils → forks**	**thinking phils, eating phils: phils**	**P: set of phils**	**x: phils**
forks		**available forks: forks**	**F: set of forks**	

Figure 2. Schema representation problem (based on Reisig (2023*b*))

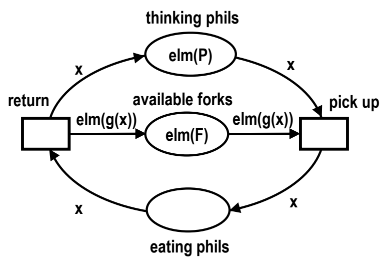
3 The *elm*-notation

Now, we want to sketch a solution for the schema representation problem. Given a signature as shown in Figure 3, we define different symbols, which describe aspects of our system. For example, there are *phils* and *forks* representing any set of philosophers and forks, respectively. The function g describes abstractly that a philosopher can have any sets of forks. Instead of having one token representing each set *P* and *F*, we now need a logical expression which allows us to generate five tokens in each place *thinking phils* and *available forks*. By this, we are able to describe each philosopher and fork by a local place for which the predicates *thinking phils* and *available forks* applies. At run level this means that each philosopher and fork is represented separately in one place. However, to consider this behavior not only in one run, but also for an arbitrary set of runs, which together describe a system model, a technique on schema level is required to deal with set symbols and to access its single elements.

Therefore, we introduce a predicate logic expression, called *elm*-notation, that allows us to generate a set of tokens from a fitting term. The *elm*-notation allows us to unfold any set symbol in a schema model and system model. In terms of the universal quantifier as provided in predicate logic, we can create as many tokens in one place as there are elements in the set for which the predicates apply. By this, we avoid an instantiation with just one token containing all philosophers (*P*) and forks (*F*). As shown in Figure 3, two domain symbols $phils$ and $forks$ are defined, which represent any set of philosophers and forks. For example, given a structure with five philosophers and forks (see Figure 1), the expressions $elm(P)$ and $elm(F)$ state that for each element of set P and set F the predicates *thinking phils* and *available forks* apply. Thus, for all philosophers (or forks), the *elm*-notation stands for the logical expression:

$$\forall p \in P : thinking\ phils(p) \tag{1}$$

Using the *elm*-notation with the function $g(x)$ as arc description, we are able to model *pick up* and *return* for any subset of forks for any subset of philosophers, allowing



a signature:

domain symbols	*function symbols*	*predicate symbols*	*constant symbols*	*variable*
phils	g: phils → sets of forks	thinking phils, eating phils: phils	P: set of phils	x: phils
forks		available forks: forks	F: set of forks	

Figure 3. Solution (based on Reisig (2023*a*))

a synchronization detached from a specific structure and its system model. Using $g(x)$, we use a function which maps a philosopher to a set of forks. Which forks are returned is not specified on schema level. This decoupling allows similar structures and any number of system models to be represented by one schema model. Furthermore, we can not only describe behavior as a sequence of occurrences of states for which typically a global timestamp is assumed. Rather a causal relationship can be described between philosophers and forks. Considering a strict partial-order instead of a strict total order, causal independence and dependency can be considered. Described by a run, for example two philosophers p_1 and p_2 could eat in parallel whereas p_1 eat with one fork and p_2 with two forks. In order to restrict the schema to the original dining philosophers problem, where philosophers can only eat with their left and right fork, constraints can be easily added for each structure.

4 Related Work

There is a plethora of other modeling techniques discussed in the literature to describe complex system behavior and dynamics. Techniques like state charts (Harel 1987), process algebras (Milner 1996, 1999) or finite automaton (Hopcroft et al. 2001) assume abstract events, abstracting away from concrete data to keep formalism better manageable. However, as the complexity of a system increases, the number of places and transitions using these techniques can grow exponentially, making the models difficult to manage, analyze, or simulate. In addition, they lack in direct support for detailed data manipulation and representation, but also representing partial-ordered runs. Petri nets are also a well-known formal approach. Various Petri net extensions like Colored Petri nets (CPN) (Jensen 1996), Predicate/ transition-nets (PrT-nets) (Genrich & Lautenbach 1981, Lindqvist 1991), algebraic-specified nets (Reisig 1991) or other high-level Petri nets (Chiola & Dutheillet 1993, Lakos 2002, Girault & Valk 2003) are discussed in literature. For example, CPNs introduce the concept of "colors", which allows distinguishing between tokens based on their data attributes (Jensen 1996). While the CPN language is based on Standard ML, it combines the strength of Petri nets and high-level programming languages to describe compactly system models in which communication, synchronization, and resource sharing play an important role (van der Aalst & Stahl 2011). Other approaches discussed are Oclets (Fahland 2009) and Proclets (van der Aalst et al. 2001, Fahland 2019). While Oclets focus on a formal process model for describing scenarios with operational semantics using Petri nets, Proclets deal with the description of system behavior based on many-to-many interactions of its data objects. However, one limitation of these approaches is that they do not know the concept of a Petri net schema. For complex systems, this can lead to very large and incomprehensible process models. In contrast, PrT-nets use predicates to describe conditions under which transitions can fire and tokens are seen as bindings of variables that satisfy these predicates. A PrT-net describes a Petri net schema with an algebra as a free variable. This allows not to represent just one system, but many similar ones. However, PrT-nets lack in generating a set of tokens from set symbols for which the corresponding term fits. A similar approach is described by Glausch & Reisig (2006). They present the concept of a Petri net schema to deal with set symbols. However, similar to a PrT-net, their approach lacks in accessing elements from set symbols itself to describe concurrent behavior, not only at run or system model level, but also on schema level.

5 Conclusion and Outlook

In this paper, we introduce the *elm*-notation to solve the schema representation problem. On the example of the dining philosophers, the idea of a universal quantifier was shown to model any set of philosophers and forks and to unfold it. By this, similar system models with their structures can be described by one schema model. We showed that the usage of the *elm*-notation is not only limited to places. Set unfolding and set aggregation can also be used as an arc label, which allows different behaviors to be taken into account depending on the underlying structure and properties. For the modeling of information systems, which deal with complex data structures, the *elm*-notation describes a promising approach. Especially, the nested hierarchical structure of business documents consisting of head and position data, like it is the case in enterprise resource planing systems (e.g. SAP), adds a layer of complexity for the description of system behavior resulting from position data. Formal approaches like HERAKLIT, which uses Petri nets, overcome the limitations of semi-formal modeling languages by representing system behavior more precisely and accurately. Semi-formal languages like BPMN or EPC often lead to ambiguities, scalability issues, and challenges in managing large systems due to lack of rigorous formalism (Lana et al. 2019). In this context, we are convinced that using the *elm*-notation, we can not only consider an arbitrary set of data objects, but also describe their behavior resulting from its single elements. For example, the processing of a purchase order can depend on the properties of its order positions like availability. Moving forward, future research endeavors will encompass additional case studies to further demonstrate the advantages of the *elm*-notation along with the provision of best practices for its effective application in modeling computer-integrated systems.

References

Chiola, G. & Dutheillet, C. (1993), 'Stochastic well-formed colored nets and symmetric modeling applications', *IEEE Transactions on Computers* **42**(11), 1343–1360.

Desel, J. (2008), From Human Knowledge to Process Models, *in* R. Kaschek, C. Kop, C. Steinberger & G. Fliedl, eds, 'Information Systems and e-Business Technologies. UNISCON 2008. Lecture Notes in Business Information Processing', Vol. 5, Springer, pp. 84–95.

Dijkstra, E. W. (1971), 'Hierarchical ordering of sequential processes', *Acta Informatica* **1**(2), 115–138.

Fahland, D. (2009), Oclets - Scenario-based modeling with petri nets, *in* G. Franceschinis & K. Wolf, eds, 'Applications and Theory of Petri Nets. PETRI NETS 2009. Lecture Notes in Computer Science', Vol. 5606, Springer, pp. 223–242.

Fahland, D. (2019), Describing Behavior of Processes with Many-to-Many Interactions, *in* S. Donatelli & S. Haar, eds, 'Application and Theory of Petri Nets and Concurrency. PETRI NETS 2019. Lecture Notes in Computer Science', Vol. 11522, pp. 3–24.

Fettke, P. & Reisig, W. (2022*a*), Breathing life into models: The next generation of enterprise modeling, *in* H. Fill, M. van Sinderen & L. A. Maciaszek, eds, 'Proceedings of the 17th International Conference on Software Technologies, ICSOFT 2022, Lisbon, Portugal, July 11-13, 2022', SCITEPRESS, pp. 7–14.

Fettke, P. & Reisig, W. (2022*b*), Discrete Models of Continuous Behavior of Collective Adaptive Systems, *in* T. Margaria & B. Steffen, eds, 'Leveraging Applications of Formal Methods, Verification and Validation. Adaptation and Learning. ISoLA 2022. Lecture Notes in Computer Science', Vol. 13703, Springer, pp. 65–81.

Fettke, P. & Reisig, W. (2024), 'HERAKLIT', `https://heraklit.dfki.de/`. Accessed: 27.03.2024.

Genrich, H. J. & Lautenbach, K. (1981), 'System modeling with high-level Petri-nets', *Theoretical Computer Science* **13**(1), 109–136. Special Issue Semantics of Concurrent Computation.

Girault, C. & Valk, R. (2003), *Petri Nets for Systems Engineering. A Guide to Modeling, Verification, and Applications*, 1 edn, Springer, Berlin, Heidelberg.

Glausch, A. & Reisig, W. (2006), How Expressive Are Petri Net Schemata?, *in* 'ICATPN', Vol. 4024 LNCS, pp. 201–220.

Harel, D. (1987), 'Statecharts: A visual formalism for complex systems', *Science of Computer Programming* **8**(3), 231–274.

Hopcroft, J. E., Motwani, R. & Ullman, J. D. (2001), *Introduction to Automata Theory, Languages, and Computation*, 2 edn, Addison-Wesley.

Jensen, K. (1996), *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Volume 1*, 1 edn, Springer, Berlin, Heidelberg.

Lakos, C. (2002), The Challenge of Object Orientation for the Analysis of Concurrent Systems, *in* J. Esparza & C. Lakos, eds, 'Application and Theory of Petri Nets 2002. ICATPN 2002. Lecture Notes in Computer Science', Vol. 2360, pp. 59–67.

Lana, C. A., Guessi, M., Antonino, P. O., Rombach, D. & Nakagawa, E. Y. (2019), 'A systematic identification of formal and semi-formal languages and techniques for software-intensive systems-of-systems requirements modeling', *IEEE Systems Journal* **13**(3), 2201–2212.

Lindqvist, M. (1991), Parametrized Reachability Trees for Predicate / Transition Net, *in* K. Jensen & G. Rozenberg, eds, 'High-level Petri Nets: Theory and Application', Vol. 13703, Springer, pp. 301–324.

Milner, R. (1996), 'Calculi for interaction', *Acta Informatica* **33**(8), 707–737.

Milner, R. (1999), *Communicating and Mobile Systems: The Pi Calculus*, 1 edn, Cambridge University Press, Cambridge.

Reisig, W. (1991), 'Petri nets and algebraic specifications', *Theoretical Computer Science* **80**(1), 1–34.

Reisig, W. (2007), 'The decent philosophers: An exercise in concurrent behaviour', *Fundamenta Informaticae* **80**(1-3), 273–281.

Reisig, W. (2023*a*), 'HERAKLIT a modeling infrastructure', `https://heraklit.dfki.de/assets/documents/mittwoch_heraklit.pdf`. Accessed: 27.03.2024.

Reisig, W. (2023*b*), 'The Essence of Petri Nets', `https://heraklit.dfki.de/assets/documents/torun_sonntag_essence.pdf`. Accessed: 27.03.2024.

Starke, P. H. (1990), *Analyse von Petri-Netz-Modellen*, Springer, Wiesbaden.

van der Aalst, W. M., Barthelmess, P., Ellis, C. A. & Wainer, J. (2001), 'Proclets: A framework for lightweight interacting workflow processes', *International Journal of Cooperative Information Systems* **10**(4), 443–481.

van der Aalst, W. M. & Stahl, C. (2011), *Modeling Business Processes. A Petri Net-Oriented Approach*, 1 edn, MIT Press, Cambridge & London.