Web-based Simulation of Production Schedules with High-level Petri Nets

Carlo Simon
Fachbereich Informatik
Hochschule Worms
Erenburgerstrasse 19, 67549 Worms, Germany
E-mail: simon@hs-worms.de

KEYWORDS

Petri Nets, Simulation Tool, Production Processes, Business Processes, Design Science Research

ABSTRACT

The paper addresses a problem, practitioners in production and logistics are faced with every day: Although several mathematical solutions including Petri net approaches exist for the simulation of complex processes, appropriate and simple to use tools do not exist. In the following, the results of a two year running research project are presented which addresses this problem. Guided by the Design Science Research approach, in this project a programming language and a programming and simulation environment for higher Petri nets has been developed, tested and evaluated. Method and tool are applied to complex simulation problems in production planning and control now. The following results had to be achieved:

1) Schedule-related data like customer orders and their prioritization can be considered now. 2) Various production strategies such as push or pull can be applied. 3) Complex data types including date and time information are supported. 4) Working time and downtime of machines can be imported from external data sources like an MES in order to calculate productivity measures like an OEE index.

The tool has been developed as a web-based system in order to support mobile use even on the shop floor. The application of language, tool and simulation results are demonstrated here introductorily at the example of a (simple) production control. The problems handled in current industry projects are more complex of course.

INTRODUCTION

The management of processes is one of the central topics in business informatics (see the remarks in (Hansen et al., 2015)): Process modeling languages are widely used to illustrate processes for analyics and optimization, as work instructions for employees, or for the specification of process aspects in software development. Popular languages for these tasks are event-driven process chains (Staud, 2006), BPMN (Allweyer, 2005), or equivalent UML languages (Randen and Fieml, 2016).

Petri nets differ significantly from these languages by their build in ability to simulate and analyze process models. For working with complex problems and models appropriate tools are needed. However, this is the core problem of Petri nets today: These tools do not exist (anymore), since the Petri net researchers of the early days developed tools mainly to answer their specific research problems but not for a broader use in engineering or software development.

The current situation is well documented looking at `https://www.informatik.uni-hamburg.de/TGI/PetriNets/tools/quick.html`, the central Petri net community website in Germany. For the 91 Petri net tools listed, the following balance can be made up:

- For 47 tools, the referenced website is no longer available or does not contain any hint on the tool anymore.
- For the next 23 tools, no development progress has been reported since 2013 or (much) earlier. Maintenance is not offered anymore and some of these tools are still MS-DOS projects.
- Another seven tools address singular research problems and can only be used by shell-scripting or as an Eclipse plug-in. They can hardly be used by others than their developers.
- In the group of the remaining 14 tools, only TimeNet supports date and time as data types (`http://www.tu-ilmenau.de/sse/timenet/`). Although the tool was still maintained until 2015, it's core bases on a project finished in 2007. The user interface is outdated and the authors do not give any hint on a mobile version or other further development.

Proceedings 32nd European Conference on Modelling and Simulation ©ECMS Lars Nolle, Alexandra Burger, Christoph Tholen, Jens Werner, Jens Wellhausen (Editors)
ISBN: 978-0-9932440-6-3/ ISBN: 978-0-9932440-7-0 (CD)

The simulation of production processes, however, needs tools which fulfill the following requirements:

- The ability to simulate timed high-level Petri nets where the tokens carry real-time information in addition to other freely definable attributes such as batch size or order priority.
- Theability to prioritize schedules, for example with regard to order priorities or the duration orders are already in progress.
- An API for accessing external data like order lists or the disposability of machines as a base for running simulations. Also there is a need to export the simulation results.
- The ability to access the application via mobile devices in order to answer optimization problems on site.
- An environment for defining processes specifications which at once generates visualizations of these processes.

This paper reports the current state of development of a novel Petri net specification language and of a web-based programming and simulation environment using this language. The illustrations shown in the following are taken from the tool. The technology base of this tool is JavaScript and SVG.

The research follows the Design Science Research approach of Hevner et al. (2004). The research project is beyond the prototype phase now and the first customers use the results from the tool for the optimization their business processes and the underlying business rules. These first results of these industry projects are reported here, too.

RESEARCH METHOD

According to (Hevner et al., 2004), there are seven guidelines regarding Design Science Research. In the following, these are briefly explained and it is shown how they are implemented within the framework of the project:

- The Design Science Research process must generate a viable artifact in form of a construct, model, method or description (*Design as an Artifact*): The artifact in this project is a grammar for the specification language and the implementation of a programming and simulation tool for this language.
- The goal of Design Science Research must be the development of a technology and / or methodology-based solution to a significant and relevant business problem (*Problem relevance*): Although several mathematical approaches for simulation of processes exist, practitioners in logistics and production often improvise using Excel or other calculation tools which are not developed for this specific problem. The approach presented here is generic and process-oriented at the same time. It facilitates rapid development of process models that can be simulated at once. Even more, real business and production data can be imported into the simulation very easily.
- The usability, quality and effectiveness of a design artifact must be evidenced by evaluation methods (*Design evaluation*): After the requirements of a simulation environment have been discussed with practitioners and a number of prototypes have been developed in the past, the simulation environment is now used in teaching and in its first customer projects. The achievements made are evaluated continuously and further requirements for future versions are derived from this.
- Design Science Research must provide a verifiable and well-structured contribution in the areas of Design Artifact, Design Foundations and Design Methodologies (*Research Contributions*): This requirement is met by the theoretical foundation based on Petri nets. An introduction to Petri nets can be found in (Baumgarten, 1996). Higher Petri net concepts used in the specification language and in the tool go back to the work of Genrich and Lautenbach (1981) and Lautenbach and Simon (1999).
- The result of Design Science Research relies heavily on the application of accurate and precise methods for creating and evaluating a design artifact (*Research Rigor*): Specification language and simulation environment are used and compared to alternative simulation environments, as shown in the introduction of this paper.
- The search for an effective artifact requires the use of available resources to achieve the desired goal while adhering to established guidelines within the considered problem environment (*design as a search process*): Three prototypes have already been developed in the past 2 years (see (Behnert, 2016), (Simon and Behnert, 2016), (Simon, 2017)). The current status can be regarded as the first productive system and is currently in use with industry partners from the fields of logistics, production, and automation. They use the simulation results to improve their business processes and business rules.
- Design Science Research and the resulting results must be disclosed and presented to both technology-oriented employees and the management (*Communication of*

Research): This requirement of the Design Science Research approach is fulfilled, since the simulation environment and the results that can be achieved with it are presented to the scientific community, students and industry partners.

FUNDAMENTAL LANGUAGE CONCEPTS AND EXAMPLES

Petri nets are bipartite graphs with places and transitions as nodes connected by directed arcs. Places are carriers of information and can be marked.

The first specification defines the behavior of a simple machine which is *preparing* to produce some product after an *init* event. After a *run* event the machine is actualle *producing* until a *stop* event occurs. The machine then turns into a *down* phase until the entire process *ends*. In case of an *error* event, the production is prohibited.

```
N Machine {
  T init;
  P preparing;
  T run;
  P producing;
  T stop;
  P down;
  T end;

  A (init, preparing);
  A (preparing, run);
  A (run, producing);
  A (producing, stop);
  A (stop, down);
  A (down, end);

  T error;
  A (preparing, error);
  A (error, down);

  P off;
  A (off, init);
  A (end, off);

  M (off=1);
}
```

This specification results in the Petri net of Figure 1. Its layout is generated automatically. Because of the marked place *off*, transition *init* is enabled indicated by its green color. If it occurs, the token is taken from place *off* and a token is put on *preparing*. This indicates the new state of the machine and the next events *run* or *error* may occur.

The attributes of the Petri net elements can easily be modified. This is demonstrated by the second example where the automatic layout is switched of and the positions of the nodes are specified with the aid of a grid. Furthermore, transition *error* is emphasized by using the color red when the transition is enabled.

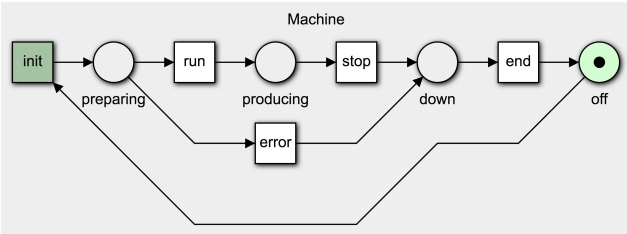

Figure 1: Petri net model of the first specification

```
N Machine(layout=false) {
  T init(col=1);
  P preparing(col=2);
  T run(col=3);
  P producing(col=4);
  T stop(col=5);
  P down(col=6);
  T end(col=7);

  A (init, preparing);
  A (preparing, run);
  A (run, producing);
  A (producing, stop);
  A (stop, down);
  A (down, end);

  T error(col=4, row=1, fillEnabled='red');
  A (preparing, error);
  A (error, down);

  P off(col=4, row=2);
  A (off, init, detour='1,2');
  A (end, off, detour='7,2');

  M (preparing=1);
}
```

The yellow color of place *preparing* indicates that this place is in conflict, i.e. either transition *run* or *error* can fire but not both of them.

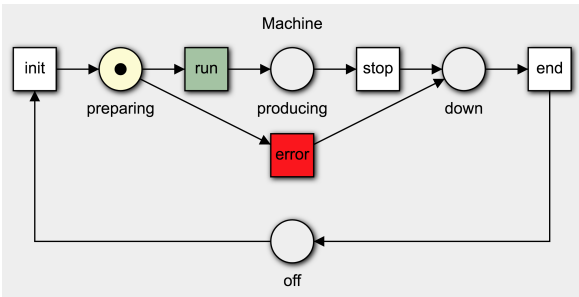

Figure 2: Petri net of the second specification

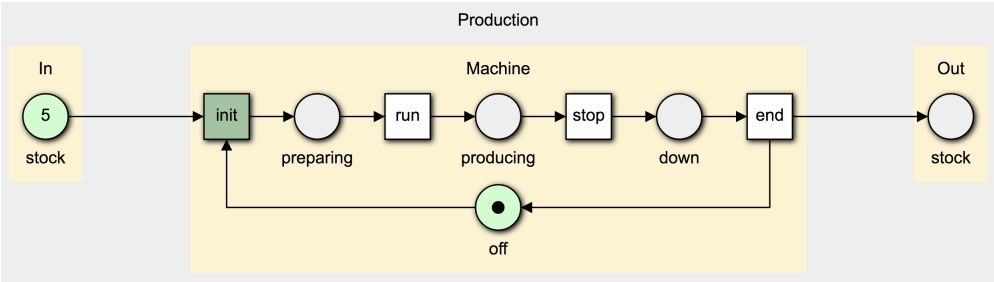

Figure 3: Petri net of two stocks and production

HIERARCHICAL MODELS

A key feature for the development of complex simulation models is the ability to group subsystems and to recycle them in another context as already stated by Stachowiak (1969). The Petri net language presented here and its specification and simulation environment allows to define hierarchical structures and to combine them as illustrated by the following example.

```
N Production {
  N In(col=1, icon='Stock') {
    P stock;
  }

  N Machine(layout=false,col=2,icon='Gear')
      {
    T init (col=1);
    P preparing (col=2);
    T run (col=3);
    P producing (col=4);
    T stop (col=5);
    P down (col=6);
    T end (col=7);
    A (init, preparing);
    A (preparing, run);
    A (run, producing);
    A (producing, stop);
    A (stop, down);
    A (down, end);
    P off (col=4, row=1);
    A (off, init, detour='1,1');
    A (end, off, detour='7,1');

    M (off=1);
  }

  N Out = In(col=11);

  A (In.stock, Machine.init);
  A (Machine.end, Out.stock);

  M (In.stock=5);
}
```

The main net *Production* consists of three subnets *In*, *Machine* and *Out* where *Out* is a copy of *In* moved into another column according to the underlying grid.

The out net is used to connect the inner nets by arcs. The name of the respective subnet is used as qualifier.

Also the initial marking of the *In* is defined in the outer net. Five tokens are put on its place which are represented as a number instead of five bullets. If the initial marking would have been defined in the definition of *In*, also this marking would have been copied to place *stock* of net *Out*.

Finally, for each of the subnets an alternative symbol is defined. The tool allows to switch between a Petri net view and a symbol view. The symbol view for this example is shown in figure 4.

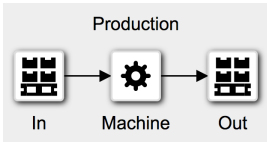

Figure 4: Alternative view on the Petri net of figure 3

MODELING OF PRODUCTION DATA

In the past, most Petri net tools have been restricted to anonymous tokens - like in the first examples of this paper - where the tokens cannot be distinguished. However then in a simulation of business or production processes important data like different orders or their priority cannot be used. Simulation is then limited to the fundamental process structure.

The Petri net language introduced here supports the definition of data types called records. A record associated with a place means that the marking of this place is restricted to tokens that correspond to this data type. This interpretation is comparable to the definition of tables in relational database systems. An example explains this concept.

The following specification begins with the definition of a record *RecordOrder* which contains an *OId* and a priority

(*Prio*), both of type *int*, and the date the order was receipt (*Receipt*) of type *date*. Both places, *oIn* and *oOut* are typed with the aid of this record definition. Moreover, the last line of the specification defines the initial marking of place *oIn*. Three tokens are put on this place.

```
N Orderreleases (layout=false) {
  R ROrder (
    OId     : int,
    Receipt : date,
    Prio    : int
  );

  P oIn(type=ROrder,col=1);
  T take(select='min(O.OId)',rcy=32,col=3);
  P oOut(type=ROrder,col=5);

  A (oIn, take, label='O');
  A (take, oOut, label='O');

  M (oIn='(1,"2018.01.24",4);
          (2,"2018.01.24",1);
          (3,"2018.01.26",2)');
}
```

Figure 5 shows the Petri net resulting from this specification. Since even a small number of individual tokens does not fit into the space given for a place, a marking is indicated by the symbol *(..)* and by colorizing the place.

Tokens in high-level Petri nets are referenced by a transition through variables annotated at the incident labels. In the example, variable *O* is used to pick up tokens from place *oIn* and to drop it on place *oOut*.

In opposite to a database, the Petri net tool presented here does not occur for sets of tokens but operates tokens in sequence. For this, a transition can be enriched by a *select*-attribute to define the order in which tokens are taken from a place. In the given example, the tokens are operated in the sequence of their *oId*.

By default, the selection condition in drawn in the center of the transition. If it does not fit into the given space, it can be moved with the aid of attribute *rcy*. Further attributes exist to modify any position of each label.

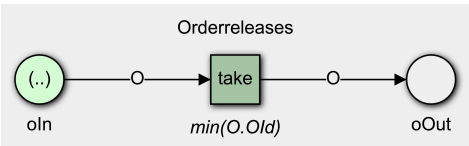

Figure 5: Typed Petri net for releasing orders

The next example demonstrates the possibility to model different selection strategies. For this record *ROrder* is reduced to two parameters: the number of an order *OId* and an identifier *CId* for the customer of this order. Place *oIn* is initially marked with three orders.

Record *RCustomer* contains two attributes: *CId* is a unique identifier for each customer and *ABC* is an indicator for her/his importance.

```
N Orderreleases (layout=false) {
  R ROrder (
    OId     : int,
    CId     : int
  );

  R RCustomer (
    CId     : int,
    ABC     : char
  );

  P oIn(type=ROrder, col=1, row=1);
  P customer(type=RCustomer,col=1,row=3);
  T take(select='min(abc)min(o)',
     col=3,row=2,rcx=16,rcy=32);
  P oOut(type=ROrder,col=5,row=2);

  A (oIn, take, label='(o,c)');
  A (customer, take, label='(c,abc)');
  A (take, customer, label='(c,abc)');
  A (take, oOut, label='(o,c)');

  M (oIn='(1,9);(2,8);(3,7)');
  M (customer='(9,"a");(8,"b");(7,"a")');
}
```

In the specification and its net shown in figure 6 a second possible way to access the tokens of a place is presented. Instead of a tuple variable which accesses an entire record at once, tuples are annotated to the arcs. This notation is used to access the single attributes of the records immediately. Label *(o,c)* of arc *oIn* → *take* references to the order number *o* and *c* to the identifier of the customer. Label *(c,abc)* of arc *customer* → *take* references to the identifier *c* of the customer and her/his priority *abc*.

Now, transition *take* is enabled, if tuples exist on both places with the same variable value for *c*. Hence a natural join of the records of places *oIn* and *customer* is built.

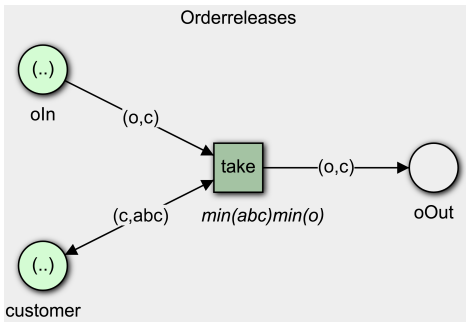

Figure 6: Releasing orders dependent on the customers' priority

Moreover this example demonstrates how to extend business rules. If the customer's priority is the main criterium for releasing an order this is expressed by putting *min(abc)* at the beginning of the selection. Since several customers might have the same priority, a second selection criterium is needed: in the example, the order number *o* is chosen.

Since the customer information must remain in the modeled system, place *customer* and transition *take* are connected by a double arc. Hence, the tuples of place *customer* are taken for the decision but are put back afterwards.

Obviously, the example is still incomplete. In a real world simulation other information like the customers' or products' names are relevant, too. The selection of these specific attributes was made to keep the example for this presentation as simple as possible.

CONDITIONAL BEHAVIOUR

The final example explains how to use transition conditions to choose the records for which a transition can occur.

```
N Orderreleases (layout=false) {
  R ROrder (
    OId     : int,
    CId     : int
  );

  R RCustomer (
    CId     : int,
    ABC     : char
  );

  P oIn(type=ROrder, col=1, row=1);
  P customer(type=RCustomer,col=1,row=3);
  T take(select='min(abc)min(o)',
      condition='abc!="x"',
      col=3,row=1,rcx=40,rcy=32);
  T leave(select='min(o)',
      col=3,row=3,rcy=32);
  P oOut(type=ROrder,col=5,row=1);

  A (oIn, take, label='(o,c)');
  A (customer, take, label='(c,abc)',
      rlx=16,rly=-16);
  A (take, customer, label='(c,abc)',
      rlx=16,rly=-16);
  A (take, oOut, label='(o,c)');

  A (oIn, leave, label='(o,c)',
      rlx=32,rly=32);
  A (customer, leave, label='(c,"x")');
  A (leave, customer, label='(c,"x")');

  M (oIn='(1,9);(2,8);(3,7)');
  M (customer='(9,"b");(8,"x");(7,"a")');
}
```

For this purpose the customer data was modified slightly. In addition to an ABC-classification of the customers a fourth classification is added which indicates suspended customers. These customers are marked with an attribute value *"x"* instead of *"a"*, *"b"*, or *"c"*.

Now, an order of a customer is taken for its release or it is left away based on the classification stored in attribute *ABC*. For this, transition *take* is extended by a condition saying that the value of variable *abc* (which corresponds to attribute *ABC*) must be different from *"x"*. For transition *leave* the selection of suspended customers is realized by using a constant in the label of *customer* → *leave*. This transition is only enabled for customers' tuples where the second attribute has value *"x"*. If it occurs, the joined order of such a customer is removed.

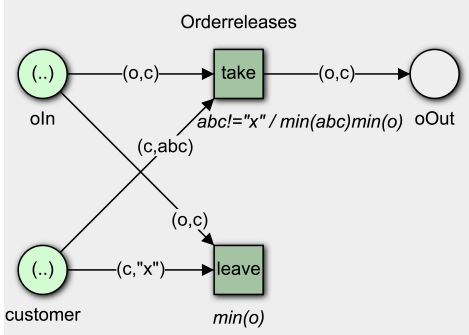

Figure 7: Conditional selection of tuples

CONCLUSIONS

What began as a small project to illustrate Petri net concepts in a Manufacturing Execution Systems course became a programming and simulation environment for Petri nets. And after a certain level of maturity was reached, the students voluntarily demonstrated the tool in their companies and reported a customers' need for this kind of a process simulation environment including a simple to use specification environment. Nowadays, the first customer projects are initiated.

Moreover, as a result of student projects and an extensive use of the tool in teaching a number of requirements have been found. The following concepts are already implemented however could not not demonstrated here in detail:

- The tool implements a layout algorithm. Its results are meaningful enough to support a rapid specification process in which users can concentrate on the internal process structure and the actual problem.
- It also implements a CSV import and export interface in order to import real business and production data

as the initial starting point for simulation and for the export of the simulation results.

- Symbols can also be used in the Petri net view and since the symbols can be animated, a meaningful process representation can be produced even for employees who are unfamiliar with process thinking.

In customers' projects the simulation environment is used to optimize business rules in logistics and production. Although some researchers assume that in such projects BPMN would be of importance, this is actually not the case. Instead of this, the mathematical foundation of Petri nets plays a more important role.

One central topic for the future development of the tool is to expand the simulation component by a controller component. In a first step, an interface for the Raspberry Pi GPIO is implemented to control machines. The further development will extend workflow management capabilities. Hence the tool will be an integrated environment for the specification, simulation and control of processes from the business level to the shop floor and back.

References

Allweyer, T. (2005): *Geschäftsprozessmanagement: Strategie, Entwurf, Implementierung, Controlling*. W3L-Verlag, Herdecke.

Baumgarten, B. (1996): *Petri-Netze: Grundlagen und Anwendungen*. Spektrum Akademischer Verlag, Heidelberg, 2nd edn.

Behnert, T. (2016): *Simulator für h¨here Petri-Netze*. Master's thesis, Provadis School of Management and Technology.

Genrich, H. J. and K. Lautenbach (1981): System Modelling with High-Level Petri Nets. *Theoretical Computer Science*, 13.

Hansen, H. R.; J. Mendling; and G. Neumann (2015): *Wirtschaftsinformatik*. De Gruyter - Oldenbourg, Berlin, 11th edn.

Hevner, A. R.; S. T. March; J. Park; and S. Ram (mar 2004): Design Science in Information Systems Research. *MIS Q.*, 28(1):75–105.

Lautenbach, K. and C. Simon (1999): *Erweiterte Zeitstempelnetze*. Fachberichte Informatik 03–99, Universität Koblenz-Landau, Institut für Informatik, Rheinau 1, D-56075 Koblenz.

Randen, H. J.and C. Bercker, van and J. Fieml (2016): *Einführung in UML: Analyse und Entwurf von Software*. Springer-Vieweg, Wiesbaden.

Simon, C. (2017): Eine Petri-Netz-Programmiersprache und Anwendungen in der Produktion. In: *Tagungsband AKWI*, eds. T. Barton; F. Herrmann; V. G. Meister; C. Müller; and C. Seel. pp. 61–70.

Simon, C. and T. Behnert (2016): Petri-Netz-Simulationen zur Theory of Constraints. In: *Tagungsband AKWI*, eds. T. Barton; F. Herrmann; V. G. Meister; C. Müller; and C. Seel. pp. 78–87.

Stachowiak, H. (1969): *Denken und Erkennen im kybernetischen Modell*. Springer, Wien, 2nd edn.

Staud, J. L. (2006): *Geschäftsprozessanalyse: Ereignisgesteurte Prozessketten und objektorientierte Geschäftsprozessmodellierung für Betriebswirtschaftliche Standardsoftware*. Springer, Berlin, 3rd edn.

AUTHOR BIOGRAPHIES

CARLO SIMON studied Informatics and Information Systems at the University of Koblenz-Landau. For his PhD-Thesis, he applied process thinking to automation technology in the chemical industry. For his state doctorate he considered electronic negotiations from a process perspective. Since 2007, he is Professor for Information Systems, first at the Provadis School of Technolgy and Management and since 2015 at the Hochschule Worms. His e-mail address is: simon@hs-worms. The tool described here can be tested on `http://www.prozesseigner.de`