Towards Simulation-Based Design of the Software Systems

Radek Kočí and Vladimír Janoušek
Brno University of Technology
Faculty of Information Technology
Bozetechova 2, 616 00 Brno, Czech Republic
{koci, janousek}@fit.vutbr.cz

Abstract—*The current trend in research of methodology of system's design highlights the quality of resulted systems, however the research mainly aims at efficiency and safety of the development process itself. The most used development methodology used models as a basic means for description of systems. Although they can allow for model transformation, the designer has to adapt the resulted code manually. It can results in inconsistency among design models and their realization and the further development or debugging by means of prime models is impossible. The paper presents an approach to the system development, which is based on the concept of simulation based design allowing to see the system as a set of models in every development stage including its deployment.*

Keywords—*simulation-based design; modeling; Object Oriented Petri Nets*

1. Introduction

The current trend in research of methodology of system's design highlights the quality of resulted systems, however the research mainly aims at efficiency and safety of the development process itself. The process of development usually uses models as a basic means for description of systems. The well-known used modeling language is UML (Unified Modeling Language) [1]. Nevertheless, the most of UML models have a static character. Their purpose is to make a conceptual design of solved problems enabling better understanding of the system design. When we wish to check dynamic properties of the developed system or to get the target application, we have to implement an executable prototype or a system according to the models in a particular programming language and framework. This classic approach often results in inconsistency among development phases—the design models become outdated and lose their value.

In the last decades, new methodologies and approaches were investigated. They are commonly known as *Model-Driven Software Development* or *Model-Based Design* (MBD) [7, 4, 3]. An important feature of these methods is the fact that they use executable models. The designer creates models and tests their correctness by simulation, so that there is no need to make a prototype. The most popular methodology is *Object Management Group's Model Driven Architecture* (MDA) [16, 20]. Thus, these methodologies use semi-formal models allowing for model transformations including code generation. Since the code has to be finalized manually with no backward transformation, it entails a possibility of semantic mistakes or imprecision between models and transformed code. Moreover, the further development, debugging and investigating of target application by means of prime models is impossible, so the significance of model has declined.

There is a lot of modeling paradigms and appropriate tools suitable for model-based design, e.g., executable UML [20], statecharts, DEVS (Discrete Event Systems Specification), Petri Nets etc. One of them is a formalism of Object Oriented Petri Nets and associated tool called PNtalk [5, 13]. The main goal of that is to experiment with modeling and simulation techniques applied in the system design [14, 15].

The paper presents an approach to the system development, which is based on three basic concepts—development in the simulation, in-the-loop simulation, and model continuity. The paper is organized as follows. First, we describe related work. The next two chapters briefly introduces the OOPN formalism and the associated design and simulation framework *PNtalk*. The fifth chapter deals with simulation-based design concepts and shows them on the simple examples. The sixth chapter outlines an approach to testing.

2. Related Work

This work benefits from two areas. First, the modeling formalism of Object Oriented Petri Nets (OOPN) [9]. The idea of merging Petri nets and objects has been found

978-0-7695-3777-1/09 $26.00 © 2009 IEEE
DOI 10.1109/ICSEA.2009.72

IEEE computer society

and elaborated in the 1990s independently by several researchers. The probably best known issues have been introduced by Lakos [18], Sibertin-Blanc [21], Moldt [19], and Valk [22]. The system Renew was found during the research of prof. Valk and dr. Moldt. Renew supports modeling and implementation of systems by an integration of Petri Nets and Java language. It is similar to the project PNtalk which benefits from OOPN formalism and Smalltalk language.

Second, the area of system design using high-level models, simulation, hardware(software)-in-the-loop simulation and model continuity. The basic idea is similar to works and platforms such as MetaEdit System[1] or Simulink.[2] MetaEdit researches Domain Specific Modeling which allows to generate code from high-level models defined for the domain-specific language. Simulink uses modeling and simulation techniques to design especially control systems and hardware architectures. It allows for hardware-in-the-loop simulation—it means that it is possible to test designed models in a real environment. The research closest to the subject of this paper is provided by prof. Zeigler and his colleagues [8]. Their approach is based on the DEVS formalism and the model continuity concept.

3. Object Oriented Petri Nets

The formalism of Object Oriented Petri nets (OOPN) [9] combines pure object-orientation inspires by Smalltalk [6] with high-level Petri nets. Petri nets allow to describe properties of the modeled system in a proper formal way and the object-orientation brings structuring and a possibility of net instantiation. This formalism is associated with a modeling and simulation framework called PNtalk [5, 13].

Object Oriented Petri Nets (OOPN) consist of Petri nets organized in classes. Each class consists of an object net and a set of dynamically instantiated method nets. Furthermore, the OOPN class defines a set of synchronous ports, a set of negative predicates, and a set of message selectors corresponding to its method nets and ports.

The object net consist of places and transitions. Each transition has conditions (i.e., inscribed testing arcs), preconditions (i.e., inscribed input arcs), a guard, an action, and postconditions (i.e., inscribed output arcs). The object net is created immediately the new instance is created. Thus, the object net describes possible autonomous activities of the object (it is possible to talk about *the object net* and *the instance* in the same matter).

The method net is similar to the object net but it has a set of parameter places and a return place. Places of the object net are accessible for the transitions of the method nets. It allows running methods to modify states of the object which they are running in. Method nets are *dynamically instantiated* by message passing specified by *transition actions*.

A token in OOPN represents either a primitive object (e.g., a number or a string) or an instance of an OOPN class. The instance consists of an instance of the appropriate object net and possibly several concurrently running instances of the invoked method nets. Message sendings and object creations are specified as actions attached to transitions. The transition execution is polymorphic — the methods which has to be invoked is chosen according to the class of the message receiver that is unknown at the compile time.

Classes can also define special methods called synchronous ports, which allow for synchronous interactions of objects. The synchronous port is a hybrid of method and transition—in order to be fired, the synchronous port has to be called (a method concept) and has to be firable (a transition concept). Synchronous port can be called only from a transition guard. The transition can be fired only if all called synchronous ports are able to fire. The synchronous port can be called with bound or unbound variables. In the case of calling synchronous ports with unbound arguments, the potential bindings of synchronous port are resolved and, if such binding exists, the arguments are bound to the resolved values. A special variant of synchronous port is *negative predicate*. Its semantics is inverted—the calling transition is firable if the negative predicate is not firable.

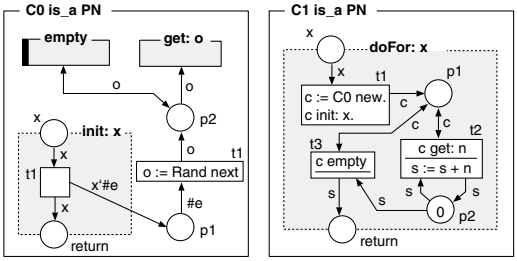

Figure 1. An OOPN example.

Object nets, method nets, and synchronous ports including negative predicates can be inherited. Inherited transitions and places of the object nets (identified by their names) can be redefined and new places and/or transitions can be added in subclasses. Inherited methods and ports can be redefined and new methods and ports can be added in subclasses.

An example illustrating the important elements of the OOPN formalism is shown in Figure 1. There are depicted

[1] http://www.metacase.com/
[2] http://www.mathworks.com/

two classes *C0* and *C1*. The object net of the class *C0* consists of places *p1* and *p2* and one transition *t1*. The object net of the class *C1* is empty. The class *C0* has a method *init:*, a synchronous port *get:*, and a negative predicate *empty*. The class *C1* has the method *doFor:*. The semantics of the method *doFor:* execution is to randomly generate *x* numbers and return their sum.

4. The PNtalk Framework

PNtalk (*P*etri *N*et *talk*) is the tool based on the formalism of OOPN [13, 5]. Its purpose is to make a framework for experiments with simulations as well as formal approaches to the system design [14, 15]. Both OOPN and PNtalk are closely associated with the Smalltalk environment. Smalltalk is the inscription language of the OOPN formalism (actions and guards are described using Smalltalk) and the PNtalk system is implemented in Smalltalk. The PNtalk system is incorporated into the other experimental tool named *SmallDEVS* [10] which is based on the DEVS formalism [23]. PNtalk uses hierarchical repositories of SmallDEVS to store OOPN classes. It is also possible to model system using together formalisms of OOPN and DEVS.

The PNtalk system architecture is based on the principles of open implementations [12], namely the meta-level architecture. The meta-level architecture distinguishes *domain model* describing developed system using appropriate domain paradigm, e.g., OOPN, and the *meta model* describing the domain model in computational environment. The meta-model introduces special objects called meta-objects allowing us to have a full control over the domain objects, its structure and behavior by means of *the meta-object protocol* (MOP). It is a crucial part of presented approach and will be demonstrate on examples.

The example of using meta-object protocol to interfacing model from Smalltalk environment is shown in the following code (the example works with models (classes) described in the Figure 1).

```
cls := rep componentNamed: 'C1'
obj := cls new.
res := obj doFor: 2.
```

The example suppose, that the hierarchical repository of stored classes is accessible via the variable *rep*. In this example, we get a metaobject representing OOPN class named *C1*. Then we create an instance of this class and send a message *doFor: 2*—it result is a sum of two randomly generated number. We can see that the two last message passing are provided in the standard form—the meta-object protocol ensures the correct processing from both Smalltalk as well as OOPN/PNtalk points of view.

The second example shows an usage of the negative predicate as follows.

```
cls := rep componentNamed: 'C0'
obj := cls new.
res := obj asPort empty.
res ifTrue: [
  obj init: 2.
].
```

We get a class *C0* and create its instance. Then we get a special metaobject representing an access to the object at the port and predicate level (`obj asPort`). Now, we can call synchronous port or negative predicate in the standard form of message passing. The result represents a state of a called port/predicate which can be tested. In our example, the predicate *empty* is true and, consequently, we will call a method *init: 2* which causes an initialization action (two random numbers will be generated and placed into the place *p2*).

5. Simulation Based Design

The Simulation Based Design (SBD) denotes, in our point of view, such an approach to the system design, which combines *model-continuity*, *incremental development in the simulation*, and *in-the-loop simulation*. These concepts should be supported by the design framework. This paper is aimed at the formalism of OOPN and the associated design and simulation framework PNtalk. Therefore the design concepts are conformed to OOPN and PNtalk.

We will demonstrate a basic concept of simulation based design on the simple example of the conference system. It comes from previously published papers [17]. We will show only a part of this example which is significant for our purpose.

5.1. Basis of the design method

The important idea behind the presented approach is *model continuity*. It makes the tendency towards an elimination of generating the source code from models [11, 14]. The system is developed incrementally, models are being improved and are simulated in each step (*incremental development in the simulation*). It is possible to simulate external components to test the system functionality. The next step is to exchange simulated components for their real realization and to test developed system in the real conditions (*in-the-loop simulation)*. Finally, the key part, in particular the control of the system logic, is kept in the system as its integral part. For example, we are able to develop the conference system in the OOPN formalism,

the user interface in Smalltalk (using, e.g., *SeaSide framework*), and to get functional application by their integration.

The methodology, like other ones, starts with analysis of the problem domain. It is possible to use, e.g., the UML diagrams (use case diagrams etc.) It helps to identify basic actors, functions, and objects of the designed system. Contrary to, let us say classic methods, the *Simulation Based Design* uses formal and simulation formalism (such as OOPN) to describe the system structure and its behavior. These models are regular part of the designed system, the system is always seen as a set of formal models in each phase of the system life cycle including its deployment.

The presented methodology uses several terms (they are not new ones, but we outline their meaning in the methodology).

- *Actor* is an object (subject) which works with the system and should have its abstract representation in the system.
- *Persistent object* is an object which is identified to be persistent in the system. The persistent object has its state and offers only a protocol to get or to modify this state. It has no own activity, i.e., it has no active role in the system.
- *Role* is a special object representing a special set of behavior of the persistent object. Each role defines at least one *task* (the main task) and can define more sub-tasks. In the case of using OOPN as the design language, the main task is modeled by object net, the sub-tasks by method nets.

We often need to have special objects playing an active or control role. This role does not have to be associated with the specific persistent object. The activities of that objects are described by means of tasks, analogous to roles. Therefore, when we will talk about task, it means the special object having control role. When we will talk about task of role, it means the behavior of the specified role.

5.2. The design process

Let us apply the previous classification to our example. We can identify *the author* and *the reviewer* as actors ans *the paper* and *review* as persistent objects. The author or the reviewer are subjects having the same basis—both are persons playing different roles in the system. Moreover, one person can play a role of author and reviewer together. So we can classify the new persistent object *member* and two roles *author* and *reviewer*. The persistent objects can be modeled by used formalism of OOPN or by means of other language (e.g., Smalltalk).

Now, let us analyze a special (control) task of the login activities. This task is modeled as an object net which is instantiated whenever the new user connects to the application. This net allows for two basic operations—to login existing user and to register new one (see the Figure 2).

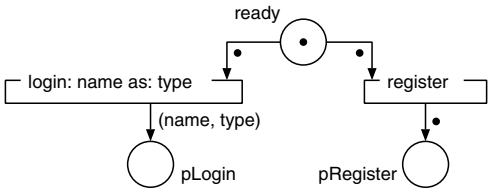

Figure 2. The login/register switch task.

First, we will pay an attention to the login process (see the Figure 3). The sequence of activity is login, verify user, and logout. If the user verification is successful, the net describing the appropriate user's role is created and placed into a place *verifiedUser*. If the user verification failed, the net state is moved back into the start marking so that the user can try to login or register again.

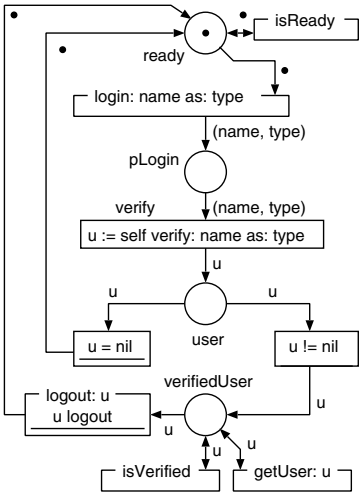

Figure 3. The login task.

The activity is modeled as a sequence of transitions or synchronous ports. While the transition firing is conditioned only by its input places, the synchronous port has to be called out, analogous to the method concept, and has to be firable, analogous to the transition concept. The transition models an internal event and the synchronous port models an event synchronized with some external event. As an example we may mention events *login:as:*, *register*, and *logout:* which are modeled by means of synchronous ports. To execute this kind of events, it has to be called from the nets surroundings. The event *verify* is modeled by means of transition because it is an internal activity of

the net. Of course, it can call some other methods or synchronous ports, but its execution is not conditioned by an external initiative.

5.3. Model continuity

Since the models can be a part of a target system, we have to ensure an interoperability of models and other programming environment. The current OOPN interpreter is implemented in Smalltalk, we are able to communicate with models from the Smalltalk environment. We will show such an interoperability on the example of a user login. Let us have a user web-based interface implemented in a Smalltalk framework. If the user connects this system for the first time, the new login/registration task is created (the object net is instantiated). The state of that object changes as the user is working.

The login task can be in following states—*ready* and *verified*. Since we need to test these states, we have synchronous (or negative) ports in this net (see the Figure 3). Now, we are able to make decision about what information will be shown as a web page based on the net's state. The Smalltalk method which ensures that can look like following code:

```
pLogNet := logNet asPort.
res := pLogNet isReady.
res ifTrue: [ ^ self welcomePage ].
res := pLogNet isVerified.
res ifTrue: [ ^ self mainPage ].
```

If the state is *ready* (or *verified*), the page described by a method *welcomePage* (or *mainPage*) is generated. Let us go to investigate the *welcomePage*. It returns a web page containing forms for user login and user registration. These activities have to correspond with external events described in the login task. When the user fill in the login form and press the button, the following method of web user interface is called:

```
pLogNet := logNet asPort.
res := pLogNet login: name as: #author.
res ifTrue: [
  res waitUntil: #(#isReady #isVerified).
  res perform.
].
```

The method tries to login a user as an author (for simplicity). It tests if the synchronous port *login:as:* is firable (if not, it implies that the system is in inconsistent state and has to be fixed). If this port is firable, we perform it. This calling is understood as an external event and the login task starts to execute—the user information will be verified and the login task's state changes to the state *verified* or the state *ready* (it depends on the verification result).

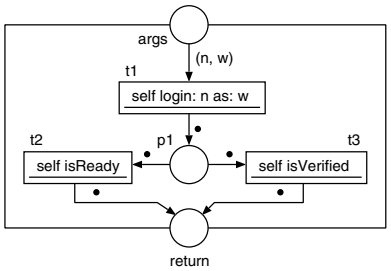

Figure 4. The generated method of wait-until.

Since we have to wait until we will know a new state, we have to declare it using a call *waitUntil:* of the metaobject protocol. Of course, we can achieve the same behavior by calling method (because the method calling is synchronous and it is possible to declare a waiting for a specific state), but we would have to (a) implement it and (b) implement different methods for different situations. In fact, the presented approach is transformed into the appropriate net (see the Figure 4), which is then called. But this is done automatically inside the metalevel.

6. System Testing and Analysis

Since we are talking about Model and Simulation Based Design, it is advisable to talk about Model Based Testing [2]. This approach supposes that test cases are derived from a model that describes some aspects of the system under test. Automatic test generation requires a language having a formal semantics, e.g., a semantics based on Finite State Machine (FSM). The semantics of an FSM based language describes translation mechanisms from a specification written in that language into a corresponding FSM describing the behavior of the tested system. The resulted representation is then analyzed and simulated (tested).

Petri nets can be used in a way similar to other formal paradigms such as FSM. Nevertheless, its special variant, Object Oriented Petri Nets (OOPN), can be used directly with no need of transformations. We just create test nets and are then able to automated test different properties in cooperation with the metalevel architecture. In the system development in simulation, we have to distinguish the system run and the simulation run. The simulation run allows to define and use special elements, e.g., we can replace a metaobject of the place by its special variant collecting statistics data. The implementation of that place is less efficient that the order one, but it is no problem in the simulation run. We can also use simulated components instead of its real (software) variant.

If we have the system described in formalisms such as OOPN, we can prepare scenarios of behavior as a workflow of transitions and synchronous ports. Then we provide a simulation run for each such a scenario. We are able to analyze, e.g., a maximum number of waiting users, the minimal and maximal time which the object spends with a waiting, etc.

7. Conclusion

The paper has described the way of the system development based on the formal models and simulation techniques. The formalism of Object Oriented Petri Nets was shown as one of possible formalisms which can be applied in the field of software engineering. The presented approach uses three crucial techniques—development in the simulation, model continuity, and in-the-loop simulation. It allows to use models as basic design and programming means. It is possible to deploy models into the target system, to test models is a real software environment, or to debug the system on the model basis. Thus, the system is always seen as a set of models and other components implemented directly in a particular language. The further research will be aimed at the efficiency of simulation techniques and at the tool support for simulation (model) based testing.

Acknowledgement *This work was supported by the Czech Grant Agency under the contracts GA102/07/0322, GP102/07/P306, and Ministry of Education, Youth and Sports under the contract MSM 0021630528.*

References

[1] J. Arlow and I. Neustadt. *UML and the Unified Process: Practical Object-Oriented Analysis and Design*. Addison-Wesley Professional, 2001.

[2] P. Baker, Z. R. Dai, J. Grabowski, O. Haugen, I. Schieferdecker, and C. Williams. *Model-Driven Testing: Using the UML Testing Profile*. Springer-Verlag, 2008.

[3] M. Broy, J. Gruenbauer, D. Harel, and T. Hoare, editors. *Engineering Theories of Software Intensive Systems: Proceedings of the NATO Advanced Study Institute*. Kluwer Academic Publishers, 2005.

[4] N. A. R. Center. Robust Software Engineering. http://ti.arc.nasa.gov/ase/index.html, [June 12, 2009].

[5] M. Češka, V. Janoušek, and T. Vojnar. PNtalk - A Computerized Tool for Object Oriented Petri Nets Modelling. In *Proceedings of the 5th International Conference on Computer Aided Systems Theory and Technology – EUROCAST'97*, pages 229–231. Las Palmas de Gran Canaria, ES, 1997.

[6] A. GoldBerk and D. Robson. *Smalltalk 80: The Language*. Addison-Wesley, 1989.

[7] J. Greenfield, K. Short, S. Cook, S. Kent, and J. Crupi. *Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools*. Wiley, 2004.

[8] X. Hu, N. Ganapathy, and B. P. Zeigler. Robots in the Loop: Supporting an Incremental Simulation-based Design Process. In *IEEE International Conference on Systems, Man, And Cybernetics*. IEEE Computer Society, 2005.

[9] V. Janoušek. PNtalk: Object Orientation in Petri nets. In *Proc. of European Simulation Multiconference ESM'95*. Prague, CZ, 1995.

[10] V. Janoušek and E. Kironský. Exploratory modeling with smalldevs. In *Proc. of ESM 2006*, pages 122–126. EUROSIS, 2006.

[11] V. Janoušek and R. Kočí. PNtalk: Concurrent Language with MOP. In *Proceedings of the CS&P'2003 Workshop*. Warsaw University, Warsawa, PL, 2003.

[12] V. Janoušek and R. Kočí. Towards an Open Implementation of the PNtalk System. In *Proceedings of the 5th EUROSIM Congress on Modeling and Simulation*. EUROSIM-FRANCOSIM-ARGESIM, Paris, FR, 2004.

[13] V. Janoušek and R. Kočí. PNtalk Project: Current Research Direction. In *Simulation Almanac 2005*. Faculty of Electrical Engineering, Praha, CZ, 2005.

[14] V. Janoušek and R. Kočí. Towards Model-Based Design with PNtalk. In *Proceedings of the International Workshop MOSMIC'2005*. Faculty of management science and Informatics of Zilina University, SK, 2005.

[15] V. Janoušek and R. Kočí. Simulation and design of systems with object oriented petri nets. In *Proceedings of the 6th EUROSIM Congress on Modelling and Simulation*, page 9. ARGE Simulation News, 2007.

[16] A. Kleppe, J. Warmer, and W. Bast. *MDA Explained: The Model Driven Architecture – Practice and Promise*. 1st edition. Addison-Wesley Professional, 2003.

[17] R. Kočí and V. Janoušek. System design with object oriented petri nets formalism. In *The Third International Conference on Software Engineering Advances Proceedings ICSEA 2008*, pages 421–426. IEEE Computer Society, 2008.

[18] C. A. Lakos. From Coloured Petri Nets to Object Petri Nets. In *Proceedings of the Application and Theory of Petri Nets*, volume 935. Spinger-Verlag, 1995.

[19] D. Moldt. OOA and Petri Nets for System Specification. In *Application and Theory of Petri Nets; Lecture Notes in Computer Science*. Italy, 1995.

[20] C. Raistrick, P. Francis, J. Wright, C. Carter, and I. Wilkie. *Model Driven Architecture with Executable UML*. Cambridge University Press, 2004.

[21] C. Sibertin-Blanc. Cooperative Nets. In *Proceedings of Application and Theory of Petri Nets*, volume 815. Springer-Verlag, 1994.

[22] R. Valk. Petri Nets as Token Objects: An Introduction to Elementary Object Nets. In *Jorg Desel, Manuel Silva (eds.): Application and Theory of Petri Nets; Lecture Notes in Computer Science*, volume 120. Springer-Verlag, 1998.

[23] B. Zeigler, T. Kim, and H. Praehofer. *Theory of Modeling and Simulation*. Academic Press, Inc., London, 2000.