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 This classic approach often results in inconsistency among development phases—the design models become outdated and lose their value.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
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 and Simulink. 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.](http://www.mathworks.com/)

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...
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 pred-
icate $empty$. 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 (Petri Net talk) is the tool based on the formal-
ism of OOPN [13, 5]. Its purpose is to make a frame-
work 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 en-
vironment. 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 hierar-
chical repositories of SmallDEVS to store OOPN classes.
It is also possible to model system using together for-
malisms of OOPN and DEVS.

The PNtalk system architecture is based on the princi-
ples of open implementations [12], namely the meta-level
architecture. The meta-level architecture distinguishes do-
main model describing developed system using appropri-
ate domain paradigm, e.g., OOPN, and the meta model
describing the domain model in computational environ-
ment. 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 interfac-
ing model from Smalltalk environment is shown in the fol-
lowing code (the example works with models (classes) de-
scribed in the Figure 1).

```apl
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 exam-
ple, 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 develop-
ment 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 as-
associated design and simulation framework PNtalk. There-
fore 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 sys-

tem. 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 elimi-
nation 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 de-
velopment 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.](image)

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.](image)
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:

```smalltalk
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:

```smalltalk
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.](image)

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