Open Implementation of the Simulation Framework

Radek Kočí *
koci@fit.vutbr.cz

Abstract: Modelling and simulation make a wide area of the computer science dealing with research on real world by means of its abstractions and experiments on these abstractions. To describe different aspects of modelled world we can use different paradigms and formalisms. The author's direction of interest has been Object oriented Petri nets (OOPNs) and their usage to modelling, simulations, and prototyping of complex systems. In course of time this interest has converged to the simulation framework intended not only to modelling and simulation of complex and parallel systems but also to making prototypes of those systems. The needs of the simple and clear controlling of features above that complex applications is still growing up. To ensure these needs the "open" approach which proposes some features to control the internal representations has been introduced. The foundation to expression of open implementation is to provide a general implementation framework making easier design and using open implementations.

Keywords: open implementation, reflectivity, metaobject protocol, Smalltalk, object oriented Petri nets

1 Introduction

Modelling and simulation make a wide area of the computer science dealing with research on real world by means of its abstractions and experiments on these abstractions. Modelling generally consists of two levels: the modelling of static structures and the modelling of dynamic aspects (behavior) of the world. Simulation can be understood as interactions of those structures based on the defined behavior. To describe different aspects of modelled world we can use different paradigms and formalisms. The author's direction of interest has been Object oriented Petri nets (OOPNs) and their usage to modelling, simulations, and prototyping of complex systems. OOPNs have been developed by our research group at Brno University of Technology and they benefit from the features of Petri nets (formal nature, suggestive description of parallelism, theoretical background) as well as object-orientedness (making structured organization, a nature creation of substructures instances, etc.) There has been realized the PNtalk project during the OOPNs evolution whereof the Petri nets and the Smalltalk system has been interconnected continuously and transparently. In present time, this interconnection forms a basis level of OOPNs thereby it becomes to be not only the means to modelling and simulation of complex and parallel systems but also the means to making prototypes of those systems.

Modern systems (and we may see the new simulations systems are modern too) become to be more and more complicated and they deal with more complex and flexible environments. It implies the needs of the simple and clear controlling of features above that complex applications is still growing up. Very essential motivation to that needs is a request to have some means to easy (perhaps even automatic) configuration and adaptation of systems to various application needs.

* Brno University of Technology, Faculty of Information Technology, Božetěchova 2, 612 66, Brno, Czech Republic

At resolving problems it is very often divided into several parts whereas these parts can be detachedly each other in less or more degree. However, a total separation like the one provided by ”black-box” abstraction can not be suitable for resolving such complex systems. Hence, the ”open” approach which proposes some features to control the internal representations has been introduced [11, 13].

The foundation to expression of open implementation is to provide a general implementation framework making easier user's goals, whether their intention is to design or to use open implementations. One of that frames (and experiences show that one of the most successful) is metalevel architecture linked to metaobject protocol (MOP). MOP provides a solution based on object orientation which can be integrated to standard development processes in simple way. It is very wide invoked in modern operating systems and languages, where it offers an elegant and uniform issue to programming and the mind based on reflect concepts.

The paper is organized as follows. Firstly, we introduce important features of OOPNs and PNtalk system. Then we describe the open implementations approaches and meaning. Consequently we take an attention to using open implementation concepts in simulation framework. Finally, we summarize current research and its development for the future.

2 OOPNs in simulations

Before dealing with OOPNs and their reference to simulation system, let we question about what is PNtalk. PNtalk is the project based on OOPN serving like an attempt implementation and checking on a possibility to use Petri nets in system design, simulation, and prototyping. Thus, we can take a look at PNtalk from several points of view. PNtalk is the language for describing models based on OOPNs; it is the system for the simulation, analyze, and verification of those models; it can be the technology of an object-oriented design, model, and prototyping of the systems. We try to describe the OOPNs and their features that are important for the PNtalk system architecture. Our description will reflect OOPNs defined in [4].

PNtalk and the associated OOPN formalism are characterized by a Smalltalk-like object orientation enriched by concurrency and polymorphic transition execution, which allow message sending, waiting for and accepting responses, creating new objects, and performing primitive computations. OOPNs are based on viewing objects as active servers[1] that offer reentrant services to other objects. Services provided by objects as well as independent activities of particular objects are described by means of high-level Petri nets consisting of inscribed places, transitions, and arcs.

An example illustrating the OOPN formalism is shown in Figure 1. As it is depicted in Figure 1, a place can be inscribed by an initial marking (a multiset of objects) and an initial action (allowing a creation and initialization of complex objects to be initially stored in the place; not shown in the Figure 1). A transition can have a guard restricting its firability and an action to be performed whenever the transition is fired. Finally, arcs are inscribed by multiset expressions specifying multisets of tokens to be moved from/to certain places by the arcs associated with a transition being fired.

When some object calls for some service of another object, videlicet the object (sender) sends a message to another object (receiver), new process is created in the receiver. This process can be understood the instance of the service description (in object-oriented terminology, the method is evoked). The process is characterized by the sequence of the transition executions,

[1] The active server means the object can have own activity which starts as soon as the object is created (it is called the *object net* in OOPNs).

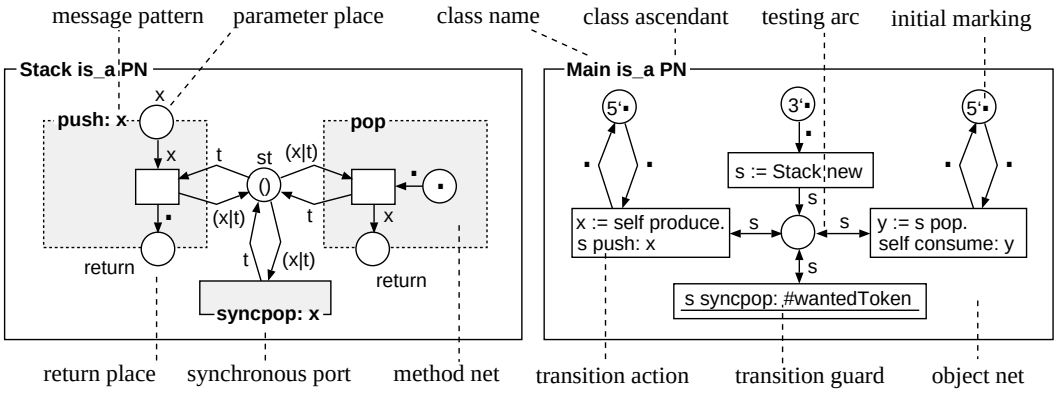

Fig. 1: An OOPN example (*Main*'s methods *produce* and *consume* are not shown).

however the notion *process* can not necessary take notice like, for example, a Smalltalk process. We can suppose each transition contains the most one message sending. If some transition sends a message to another object, then it has to wait for response and the processing of this transition is suspended[2] (not the process). As it results from the Petri nets definition, each process can have several transitions that are independent each other, and consequently they can do concurrently. Thus, there is a parallelism within the OOPNs process. This is probably the most important feature of OOPNs from the system architecture point of view.

3 Open implementations issue

Open systems (or open implementation of systems) can be understood and defined by various ways. Because of it is an essential property of theoretical cogitation and practical aspects presented in this paper, we try to outline and to explain this notion from several point of view. The most important one are these: why we deal with it (significant point of view) and how to make a framework intended for building the open systems based on it (realization point of view).

Recent trend in modern systems for complex application support (e.g. operating systems) is not only to allow applications to use services offered by the given system, but also to offer means to control how these services are provided and processed. This trend can seem to be contrary to more traditional approach – black-box abstraction. It says some abstraction (object) should expose its functionality but hide its implementation. Black-box abstraction has many attractive qualities and brings a possibility of portability, reusing or simplicity of the design process. Nevertheless we can even find disadvantages linked with choosing well-fitted implementation in reusing, with adapting chosen part to the used environment, or with evolution of application during its life-time. The open implementation principles offer a solution of problems above. It is needed to remark that these principles should be understood rather like the framework intended for more flexible design and using of black-boxes[3].

The most use principle of open implementations is *metalevel implementation* including its *metaobject protocol*. The vocable *meta*[4] marks higher sphere of the control and de facto de-

[2] Take a notice the message sending is rather the request for message processing by the receiver and the receiver processes the message within a newly created process.

[3] Thus, it is not good idea to reprobate the black-box abstraction. However, in some cases there is useful to have mechanisms to partly evade the black-box concepts.

[4] The vocable meta comes from Greek and means something like *after* or *behalf*. In our conception it can be understood as a signification of something what stays behind an object and reflects (or describes) its features and properties.

scribes information about information. Suppose the application is represented by a set of objects whereas the behavior of objects is described by classes. Now we need to distinguish a control sphere from a regular sphere. To reach this situation we can imagine two different levels of control flow: *domain level*[5] and *metalevel* [11].

As an example we may name the Smalltalk system [2]. Each class has its metaclass which has the same meaning like the class for regular objects. Metaclasses control the class behavior and classes control the object behavior. Thus, Smalltalk has two levels of controlling (it implies there is one metalevel) – the one for object controlling and the one for class controlling.

Let us define following terms:

domain level *Application level describing solved problems by means of available formalisms. On this level the evolution of domain objects (i.e. objects having a direct connection to real solved problem) acts.*

meta level *Metalevel is generic notation making a space to special (control, implement) metaobject evolution.*

metaobject protocol *Object oriented interface allowing objects and metaobjects to communicate each other.*

4 Open implementations in the simulation framework

4.1 The metalevel architecture

The metalevel architecture of the simulation framework is based on three following levels (the Figure 2 illustrates them):

domain level: The domain level is represented by domain classes[6] and their instances (i.e. objects). In the domain level point of view it is a classic object oriented approach to model creation or to problem solving. In the system point of view these objects are *abstract* which do not physical exist and they are represented by their metaobjects. In the Figure 2 the domain objects are shown at the down having the notice *abstract*. The dashed lines with filled triangle represents a relationship instance–class. The dashed lines with hollow triangle represents a relationship domain object–metaobject.

meta level: The meta level above domain level is represented by a set of instances of special (meta)classes. The basic ones are *PNClass* (describes the OOPNs classes behavior) and *PNMetaObject* (describes the live objects behavior). In the Figure 2 the meta objects are shown at the down above the domain level objects.

meta meta level: The meta level above meta level. It is represented by special (meta)classes introduced in previous item. Those (meta)classes are instances of their metaclasses in Smalltalk. There is just one object on the meta level for each object on the domain level (and vice versa). For each *type* of domain object there is just one class on the meta meta level. All meta objects are instances of appropriate object on the meta meta level for given type of the domain object.

[5] In literature the term *basic level* is rather use. We will still use a term domain level because it better represents a foundation of this level in my opinion – i.e. the reflection of solving problems.

[6] The term *domain class* represents a class consisting of methods describing a part of solved problems. The methods can use any paradigms available in the simulation framework. The prime paradigm is OOPNs and whole framework is designed in accordance to generalized principles of the OOPNs evolution.

The system architecture introduces a new meta level between the domain conception of a class and own (implementation) system Smalltalk. On this meta level we must distinguish two kinds of meta objects: the metaobject serving a domain class and the metaobject serving a domain object (i.e. an instance of domain class). While the metaobject of the first kind describes *a structure* of the domain class and it also defines the ways of manipulation with the domain class, the metaobject of the second kind just describes *a computational behavior* of objects (thus instances of domain classes). This ”live” part of the meta level is characterized by three features that are different from the Smalltalk (and also general) conception of meta levels:

– the domain object *is not* an instance of the meta object. Thus, the meta object does not say *what* the object is to do even what is its structure but it only *reflects* the object aspects by appropriate patterns.
– the meta object is *actor*. It consists of the process serializing all accesses and requirements to the object.
– the meta object controls the operations on the domain object by virtue of *control messages*.

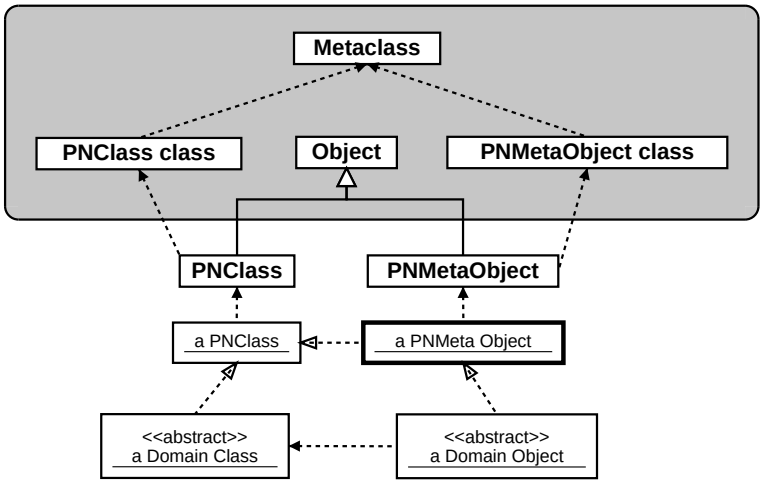

Fig. 2: Open simulation system metalevels – basic overview

4.2 The base of system

The system conception is inspired with experiences and formal definition of OOPNs. The evolution principles of OOPNs can be generalized for substantial part of the process and event oriented computation.

Thread: Thread is basic element in the model evolution. In OOPNs it is represented by fired transition and de facto the thread is the least process unit.

Process: Process is understood as an instance of method which has ensued in depending on method invocation. The process consists of a set of threads. We suppose the process contains at least one thread.

Object: The object represents live domain object in the system (e.g. the OOPNs object). It consists of a set of processes.

World: World represents a group of objects collaborating in the simulation. There always exists at least one (default) world.

Event: The model evolution is controlled by events. There can arise several kinds of events. Threads, processes, objects, and worlds evolve depending on the causality and the kind of events.

The system is organized into following hierarchy: $World$ controls $Object$; $Object$ controls $Process$; $Process$ controls $Thread$.

4.3 The base definitions

Now let us define the essential aspects of the simulation system. The definition is very useful for outlining the basic principles of model evolution and also the basic principles of *the system reflection*. Each defined structure is linked to operations on it; due to limited space of the paper we will abstract these operations.

Definition 1 *Process is defined as a structure*

$$P = (calendar,\, threads,\, events,\, \ldots)$$

where

- $calendar$ *matches events linked to time for this process*
- $threads$ *is a set of threads in the process*
- $events$ *is a set of events in the process*
- $\ldots$ *is a list of other components of process. These components depend on using paradigm.*

How we can see the proces definition has two parts: the fixed part and the variable part. The fixed part must be satisfied for each paradigm being to use while the variable part depends on used paradigm. In case OOPNs the variable part is $transitions$ and $places$. So we define further terms:

- $P_{PN} = (calendar,\, threads,\, events,\, transitions,\, places)$ as OOPNs process
- Π_{PN} is a set of all OOPNs processes
- $\Pi = \Pi_{PN}$ is a set of all processes in simulation framework

Definition 2 *Process binding is defined as a pair*

$$thread \rightarrow process$$

where

- $thread$ *represents depending thread*
- $process$ *represents evoked process*

To explain the term *process binding* let us imagine following train of thought. Let us have objects A and B, the process P_A like process of object A and the event e_A like an event in the process P_A. The event e_A represents the sending of message $message$ to the object B. It evolves the process P_B in the object B. The event e_A is a part of thread t_A. Owing to message processing the event e_A causes the thread t_A is suspended until the process P_B is terminated. It follows the thread t_A depends on the process P_B. Then we will write $t_A \rightarrow P_B$.

Definition 3 *Domain metaobject is defined as a structure*

$$Domain\,Meta = (Slave,\, ev\,P,\, dp\,T,\, queue,\, class,\, activism)$$

where

- $Slave$ *is direct slave object*
- $ev\,P$ *is a set of evoked processes*

– dpT *is a set of depending threads*
– $queue$ *is a queue of delegated messages*
– $class$ *is a reference to the domain class*
– $activism \in \{true,\ false\}$ *determines an object activity*

The domain metaobject is metaobject serving domain objects (i.e. instances of domain classes). It consists of slave object (it is necessary to ensuring serialization of requirements and simpler manipulation with objects), sets of $ev\,P$ and dpT (thus the metaobject knows the process dependings), the queue of incoming messages to be serialized, and the information about a domain class and activity of object.

Definition 4 *Slave metaobject is defined as a structure*

$$Slave = (LP,\ processes)$$

where

– $processes = \{\, p \,|\, p \in \Pi \}$
– LP *is the life process of the object,* $LP \in \Pi \cup \{\varepsilon\}$*, where* ε *represents an empty process*

5 The meaning of open simulation system

The open simulation system conceptions has its disadvantages and its advantages. Firstly we will deal with advantages. The most interesting is a possibility to adapt model during its evolution to required form and a possibility to adapt expression paradigms to get better solution of modelled problems. For example, when we make decision some part of the model is to be described by process-oriented paradigm (e.g. Simula-like processes in Smalltalk code) we may derived a new structure of process type, such as

$$P_S = (calendar,\ threads,\ events)$$

and reimplement operations on this structure. Then Π_S is a set of all Simula-like processes and the set Π will be extended to $\Pi = \Pi_{PN} \cup \Pi_S$. The process holds just one thread, the event is at most one (if the thread is suspended and then resumed so the event occurs). Thus, if we keep defined structures and interfaces of operations on them we may derive new kinds of processes and consequently we may use new kinds of paradigms in the simulation system.

The further possibility to use the reflective features of open implementation of the simulation system are special simulation techniques. As an example we can take attention to nested simulation. The principles and case study was published at the Association of Simula Users Conference [5]. We may imagine the nested simulation as *the thinking models about itself*. The model derives a model on itself (that submodel can be simplified prime model, the same prime model, or other abstraction of prime model) and the submodel simulation then represents that thinking about prime model. To ensure those manipulations on models or system we can use operations from metaobject protocol, for instance *the cloning* of the simulation world or the making submodels in worlds directly.

There are two main disadvantages: the overhead of model evolution grows up; the security and the safe of evolution partially steps down. Although those facts the advantages outweighs disadvantages. Nevertheless it is very important to keep in mind possible hazards linked to extraordinary using the means that the system offers.

6 Conclusions

This paper has presented the meaning of open implementation concepts used in conjunction with open simulation framework design. It brings the key ideas of the author's Ph.D. thesis being completed this year in brief way. The thesis appears from the idea of the simulation system open implementation which converges to a lite form of an operating system serving for modelling, simulation, and prototyping complex systems. The advantages of open implementations should approve in a robust flexibility of the system enabling to adapt environment to required needs, including a possibility to change or to combine different paradigms.

This work has been done within the project CEZ:J22/98: 262200012 "Research in Information and Control Systems" and also supported by the Grant Agency of Czech Republic grants No. 102/04/0780 "Automated Methods and Tools Supporting Development of Reliable Concurrent and Distributed Systems".

Bibliography

1. M. Češka, V. Janoušek, and T. Vojnar: *PNtalk – A Computerized Tool for Object-Oriented Petri Nets Modelling*. In F. Pichler and R. Moreno-Díaz, editors, *Computer Aided Systems Theory and Technology – EUROCAST'97*, volume 1333 of *Lecture Notes in Computer Science*, pages 591–610, Las Palmas de Gran Canaria, Spain, February 1997. Springer-Verlag.
2. A. Goldberg and D. Robson: *Smalltalk 80: The Language*. Addison-Wesley, 1989
3. V. Janoušek: *Reflective Approach to Petri Net Simulation.* MOSIS'97, MARQ Ostrava, pages 209–304.
4. V. Janoušek: *Modelování objektů Petriho sítěmi (in czech)*. Ph.D. thesis at DCSE FEECS TU of Brno, 1998.
5. V. Janoušek, R. Kočí: *PNtalk - An Open System for Prototyping and Simulation*. In: Proceedings of The 28th ASU Conference, Brno, CZ, FIT VUT, 2002, p. 133-146, ISSN 1102-593X
6. E. Kindler et al.: *Special Approach to Reflective Simulation*. In: Proceedings of MOSIS 2001. MARQ. 2001.
7. R. Kočí: *Rozšiřitelný simulátor objektově orientovaných Petriho sítí (in czech).* Thesis at DCSE FEECS TU of Brno, 2000.
8. R. Kočí, T. Vojnar: *A PNtalk-based Model of a Cooperative Editor*. In Proceedings of the 35th Spring International Conference on Modelling and Simulation of Systems – MOSIS 2001, Hradec nad Moravici, Czech Republic, CZ, MARQ, 2001, p. 165-172, ISBN 80-85988-57-7
9. R. Kočí: *The PNtalk System - a Technique for Object Oriented Modelling*. In: Proceedings of XXII-Ird International Autumn Colloquium, Ostrava, CZ, MARQ, 2001, p. 151-158, ISBN 80-85988-61-5
10. R. Kočí, Z. Rábová: *The PNtalk System and Interoperability*. In: Proceedings of International Conference MOSIS '02, Ostrava, CZ, MARQ, 2002, p. 73-80, ISBN 80-85988-71-2
11. P. Maes: *Computational reflection*. Technical report, Artifical Inteligence Laboratory, Vrije Universiteit Brusel, 1987
12. Hidehiko Masuhara and Satoshi Matsuoka and Takuo Watanabe and Akinori Yonezawa: *Object-oriented concurrent reflective languages can be implemented efficiently*. In: Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA), SIGPLAN Notices, volume 27, number 10, ACM Press, New York, NY, 1992, p. 127–144
13. Renaud Pawlak: *Metaobject Protocols For Distributed Programming*. Technical report, Laboratoire CNAM-CEDRIC, Paris, 1998