Modeling and Evaluation of Software Systems with Object Stochastic Activity Networks

Mohammad Abdollahi Azgomi
Department of Computer Engineering,
Iran University of Science and Technology,
Tehran, Iran
E-mail: azgomi@iust.ac.ir

Ali Movaghar
Department of Computer Engineering,
Sharif University of Technology,
Tehran, Iran
E-mail: movaghar@sharif.edu

***Abstract—Stochastic activity networks* (SANs) are a stochastic generalization of Petri nets. SAN models have been used to evaluate a wide range of systems and are supported by several modeling tools. We have introduced *object stochastic activity networks* (OSANs) to overcome some restrictions of these models. OSANs integrate the concepts of object-orientation into SAN models. Elements of OSANs and their submodels are defined as classes. OSANs are more appropriate that most other object-oriented or high-level extensions of Petri nets for application on software systems. In this paper, we will present the definitions, behavior and an example of OSAN models. The object-orientation of OSANs and the flexibility of having functions for activities, make these models more appropriate than other extensions of Petri nets for modeling and evaluation of software systems.**

Keywords- modeling, evaluation, Petri nets, object stochastic activity networks, software performance

I. INTRODUCTION

Stochastic activity networks (SANs) [25] are a stochastic generalization of *Petri nets*. These models are more powerful and flexible than most other stochastic extensions of Petri nets including notable models such as *stochastic Petri nets* (SPNs) [21] and *generalized stochastic Petri nets* (GSPNs) [7].

SAN models have been used as a modeling formalism in several modeling tools, such as *UltraSAN* [28] and Möbius [14] and have been used to evaluate the performance and dependability of a wide range of systems.

In order to integrate the concepts of *object-orientation* (OO) into SANs and to overcome some restrictions of these models, we will introduce an object-oriented extension for SANs, called *object stochastic activity networks* (OSANs) in this paper. OSAN models are appropriate for object-oriented modeling (OOM) of software systems and can be evaluated by analytic or simulation techniques.

Primitives of OSANs and their submodels (called *super activity*) are defined as classes. Tokens of OSANs are objects of a user-defined token class. These models adapt some useful ideas and features from the existing high-level and object-oriented extensions of Petri nets.

Preliminary ideas and informal definitions of OSANs have already been introduced in [1] and [3]. In this paper, we will present their finalized graphical notations, formal definitions and an example.

This paper is organized as follows. Sec. II some related works are introduced. Sec. III mentions the motivations of this work. In Sec. IV, the informal and formal definitions, graphical notations and an example of OSAN models are presented. Finally, in Sec. V, some concluding remarks and future works are mentioned.

II. RELATED WORKS

Object-orientation has been identified as a key answer to main critiques of Petri nets. An object-oriented paradigm provides excellent concepts to model real-world problems [16]. OO concepts allow constructing models easily, intuitively and naturally.

The absence of structuring capabilities has been one of the main criticisms raised against high-level Petri nets [15]. The attractive characteristics of these nets have prompted researchers to enrich these formalisms with object-oriented features.

Proposals for the integration of OO and Petri nets are numerous and widely different. However, two major trends are as: o*bjects inside Petri nets* vs. *Petri nets inside objects* [10]. C.A. Lakos has also categorized them as: *object-based Petri net* (OBPN), *object-oriented Petri net* (OOPN) and *object Petri nets* (OPN) [24].

Different styles and dialects of OOPNs are described in the literature. A historical list of OOPNs is shown in Tab. 1. For a detailed comparison, please see [10], [24], [15] and [16].

III. MOTIVATIONS

The integration of object-orientation into SANs, which are a stochastic modeling formalism, will have a lot of advantages. To be able to define tokens as instances of user-defined classes and definition of submodels as classes and using several instances of them to construct a model will facilitate modeling software systems.

Proceedings of the International Conference
on Software Engineering Advances (ICSEA'06)
0-7695-2703-5/06 $20.00 © 2006 IEEE

IEEE COMPUTER SOCIETY

TABLE I. A HISTORICAL LIST OF OOPNS

Year	OO Extensions of Petri nets
1988	PROT nets [8]
1988	OBJSA nets [11]
1991	SimCon object model [30]
1991	Language for OO Petri net (LOOPN) [22]
1991	Concurrent OO Petri net (CO-OPN) [13]
1993	Expressive comfort [31]
1993	Cooperative objects [9]
1994	Cooperative nets [27]
1994	LOOPN++ [23]
1995	Object Petri nets (OPNs) [24]
1995	Class oriented with nets (CLOWN) [12]
1997	PNTalk (PN + Smalltalk) [32]
1998	Object nets [29]
1999	Reference nets [19]
2000	Hierarchical OO Nets (HOONets) [16]
2001	Object coloured Petri nets (OCP-nets) [20]

The existing object-oriented extensions of Petri nets are rather complicated. They lack simplicity, good graphical representations and analysis techniques, which are the key benefits of the original Petri nets. Some of the OOPNs are defined as textual modeling languages (ex. LOOPN) [22]. Most of them, such as OPNs, are behaviorally equivalent to coloured Petri nets (CPNs) [17]. Therefore, these models are not timed and stochastic and therefore, inappropriate for evaluation purposes. Another disadvantage of OOPNs is their weak tool support (comparing to SPNs and CPNs).

The aims of OSANs are as follows:

1. *Integrating OO concepts into a stochastic modeling formalism*: Those parts of OO concepts that are more useful, will be integrated into SANs. *Classes*, *hierarchy of classes and objects* and *inheritance* are the most important features that are selected.
2. *To be useful for object-oriented modeling of software systems*: The unified modeling language (UML) is the de facto standard of OOM of software systems. However, UML lack formal semantics. Therefore, UML models are inappropriate for verification and evaluation purposes. Petri nets and other formalisms have been used as a complement to Petri nets in several research projects. While, having a formalism that takes the advantages of both of the UML and stochastic Petri nets is a better solution for this purpose.
3. *To be based on "objects inside Petri nets" approach*: Tokens of OSANs are objects of a user defined token class. These objects can be manipulated by elements of OSAN models.
4. *Submodels as classes*: Submodels of OSANs can be defined as classes. Each of these classes can be instantiated with different names to compose an OSAN model. For each submodel class, some data fields and methods can be defined. These methods are used to access to the internal states (marking of places) and data fields of submodels. Therefore, submodels of OSANs are fully encapsulated. This is a key benefit of OSANs over *hierarchical SANs* (HSANs) [4].

IV. OBJECT STOCHASTIC ACTIVITY NETWORKS

In this section, we introduce object stochastic activity networks, including their elements, formal definitions, graphical notations and an example.

Our base model for the definition of OSAN models is a new definition of SANs [26] that is slightly different from their original definition [25].

A. Elements of OSAN Models

The elements of OSANs are *simple place*, *coloured place*, *instantaneous activity* and *timed activity*. The simple places hold black tokens of integer type, while the coloured places hold a list of tokens (*objects*) of a user-defined *token class*.

The input and output gates of the ordinary SANs do not exist in OSANs. These elements are encapsulated into instantaneous and timed activities. The reasons for this integration are as follows:

- Gates are not separate elements. They define the actions of a linked activity. Such a thing is not useful in OO methodology. Therefore, they are encapsulated into activities as their actions, methods or functions.
- Gates are non-standard elements, which are not found in standard extensions of Petri net.
- There are a number of ambiguities in the graphical representation of standard gates and general gates.

However, the advantages of gates, which are their flexibilities for construction of condensed models is still preserved in the definition of activities that will be explained later.

In the following paragraphs, we introduce OSAN elements and their graphical notations:

1. Simple place. A *simple place* of OSANs is similar to ordinary SANs and Petri nets. A simple place holds black tokens of *integer* type.

2. Token class. Coloured tokens that are held by coloured places are objects of a user-defined *token class*. A token class has one or more *fields* and zero or more *methods*. Methods of a token class are general functions. These methods can be called by functions of activities. It means that the treatment of tokens by activities can vary from one token to another, depending on their fields and methods. The visibility of each field or method is determined by *private*, *public* or *protected* keywords. The syntax for the token class definition is quite similar to C++ or Java. The formal definition of token class is as follows:

Definition 1. *Token class* is defined as a tuple $TC = (ID, DF_1, \ldots, DF_n, M_1, \ldots, M_m)$, where:

- ID is a unique identifier of the token class,
- $DF_1, \ldots, DF_n$ are *fields* of the token class, where $n \geq 1$ and each field is defined as a 4-tuple (FID, FT, FC, FV), where FID is a unique identifier for the field; FT is a data type; FC is a field constraint for the values to be bound to the field; and FV is the visibility of the field.

Proceedings of the International Conference
on Software Engineering Advances (ICSEA'06)
0-7695-2703-5/06 $20.00 © 2006 IEEE

IEEE COMPUTER SOCIETY

- $M_1, \ldots, M_m$ are *methods* of the token class, where $m \geq 0$ and each method is defined as a 4-tuple (*MID*, *T*, *G*, *MV*), where *MID* is a unique identifier for the method; *T* is the data type of method's return value; *G* is the function of the method; and *MV* is the visibility of the method.

3. Coloured place. A coloured place holds a *list* of tokens of a specified *token class*. To each coloured place is associated *a selection policy*, which specifies the order of tokens to be taken from the coloured place. The possible selection policies are {*NSP*[1], *FIFO*, *LIFO*, *PRI*}. If the selection policy of a coloured place is *NSP*, the list of tokens is unordered; otherwise, it is an ordered list. For *FIFO* and *LIFO* selection policies, the temporal order of tokens is used. For PRI selection policy, a user-defined order function determines the order of tokens inside the coloured place.

Both the number of tokens inside a coloured place (i.e. the size of the token list) and the values of its fields can be read in predicates and functions or can be read or manipulated by functions of activities. The following is the formal definition of coloured place:

Definition 2. A *coloured place* is defined as a triplet *CP* = (*TC, SP, OF*), where:

- *TC* is the identifier of a token class,
- $SP \in \{NSP, FIFO, LIFO, PRI\}$ is a selection policy for the place, and
- *OF* is the *order function* of the place and is defined if *SP* = *PRI*.

The graphical representation of a coloured place is shown in Fig. 1, where *TCName* is the name of a token class and *sp* is the selection policy of the place.

Figure 1. The graphical notation of a coloured place

4. Timed activity. *Timed activities* represent activities of the modeled system whose durations impact the system's ability to perform. Timed activities represent *parallelism* in the modeled system. A timed activity in OSANs is an encapsulation of *timed activity*, *input gate* and *output gate* of the original SANs. A timed activity is linked directly to some *input* and *output* places. To each timed activity is associated an activity function that behaves like functions of input/output gates.

The graphical representation of a timed activity with *m* inputs and *n* outputs is shown in Fig. 2.

5. Instantaneous activity. Instantaneous activities describe events, which occur instantaneously and are completed in a negligible amount of time. *Case probabilities* associated with instantaneous activities permit the *probabilistic modeling nondeterminacy*. An instantaneous activity in OSANs is an encapsulation of *instantaneous activity*, *input gate* and *output gate* of the original SANs. To each instantaneous activity is associated an activity function.

The graphical representation of an instantaneous activity with *m* inputs and *n* outputs is shown in Fig. 3.

Figure 2. The graphical notation of timed activity

Figure 3. The graphical notation of instantaneous activity

B. Definition of Flat OSAN Models

A flat OSAN model is composed of the above elements and some data fields and methods. Data fields of an OSAN model may be used as variables in activity functions. Methods of an OSAN model are private and local functions that can be called by activity functions.

Flat OSAN models are formally defined as follows:

Definition 3. *Object stochastic activity network* (*OSAN*) is defined as a 13-tuple *OSAN* = (Σ, *DF*, *M*, *SP*, *CP*, *IA*, *TA*, *IR*, *OR*, *C*, *F*, Π, ρ), where:

- Σ is a finite set of *token types*.
- *DF* is a finite set of *data fields*. Data fields are defined like Definition 1.
- *M* is a finite set of *methods*. Data fields are defined like Definition 1.
- *SP* is a finite set of *simple places*.
- *CP* is a finite set of *coloured places*.
- *TA* is a finite set of *timed activities*. To each timed activity, *a*, with *m* inputs and *n* outputs is associated:
  - A computable predicate, *e*, called the *enabling predicate* of *a*,

    $$e: M_I \rightarrow \{true, false\},$$

    where $M_I = M_1 \times \ldots \times M_m$, × stands for Cartesian product and M_i for $i = 1, \ldots, m$ is defined as follows:
    - If $P_i \in SP$: $M_i = N$, where *N* denotes the set of positive integer numbers.
    - Otherwise (i.e. $P_i \in CP$): $M_i = L(P_i)$, where $L(P_i)$ is the set of all possible occurrences (closure) of the token list of P_i.

[1] *NSP* is abbreviation for *no-selection-policy*

Proceedings of the International Conference
on Software Engineering Advances (ICSEA'06)
0-7695-2703-5/06 $20.00 © 2006 IEEE

IEEE COMPUTER SOCIETY

- A computable partial function, G, called the *activity function* of a,

$$G: M_I \rightarrow M_O,$$

where M_I is defined as before *and* $M_O = M_1 \times ... \times M_n$, where M_i for $i = 1, \ldots, n$, and n is the number of activity outputs and is defined as before.

- IA is a finite set of *instantaneous activities*. To each instantaneous activity, a, is associated an enabling predicate, e, and an activity function, G, that are defined as before.
- $IR \subseteq (SP \cup CP) \times \{1, \ldots, |SP \cup CP|\} \times (IA \cup TA)$ is the *input relation*[2]. IR satisfies the following conditions:
  - For any $(P_1, i, a) \in IR$ such that a has m inputs, $i \leq m$,
  - For any $a \in TA \cup IA$ with m inputs and $i \in N$, $i \leq m$, there exist $P_1 \in (SP \cup CP)$ such that $(P_1, i, a) \in IR$,
  - For any $(P_1, i, a_1), (P_1, j, a_2) \in IR$, $i = j$ and $a_1 = a_2$.
- $OR \subseteq (IA \cup TA) \times \{1, \ldots, |SP \cup CP|\} \times (SP \cup CP)$ is the *output relation*[3]. OR satisfies the following conditions:
  - For any $(a, i, P_1) \in OR$ such that a has m outputs, $i \leq m$,
  - For any $a \in (IA \cup TA)$ with m outputs and $i \in N$, $i \leq m$, there exist $P_1 \in (SP \cup CP)$ such that $(a, i, P_1) \in OR$,
  - For any $(a_1, i, P_1), (a_2, j, P_1) \in OR$, $i = j$ and $a_1 = a_2$.
- $C: M \times IA \rightarrow [0, 1]$ is the *case probability function*, where $M = M_1 \times ... \times M_n$, $n = |SP \cup CP|$ and M_i for $i = 1, \ldots, n$ is defined as before.
- $F = \{F(.|\mu, a); \mu \in M, a \in TA\}$, where M is defined as before is the set of activity *time distribution functions*, where $n = |SP \cup CP|$ and, for any $\mu \in M$, and $a \in TA$, $F(.|\mu, a)$ is a probability distribution function,
- $\Pi: M \times TA \rightarrow \{true, false\}$ is the reactivation predicate, where M is defined as before,
- $\rho: M \times TA \rightarrow R+$ is the *enabling rate function*, where M is defined as before and $R+$ denotes the set of positive real numbers.

Similar to a new definition of SANs [26], in addition to the above stochastic setting, two other settings, namely nondeterministic and probabilistic settings can be defined as follows:

Definition 4. The nondeterministic setting of OSANs, that is called *object activity network* (*OAN*) is derived from Definition 3 and is defined as an 8-tuple $OAN = (\Sigma, DF, M, SP, CP, IA, TA, IR, OR)$, where Σ, DF, M, SP, CP, IA, TA, IR and OR are defined as before.

Definition 5. The probabilistic setting of OSANs, that is called *object probabilistic activity network* (*OPAN*) is derived from Definition 3 and is defined as a 9-tuple $OPAN = (\Sigma, DF, M, SP, CP, IA, TA, IR, OR, C)$, where Σ, DF, M, SP, CP, IA, TA, IR, OR and C are defined as before.

[2] In a graphical representation, $(P_k, k, a) \in IR$ means that place P_k is linked to k-th input of activity a. P_k is said to be an input place of a.

[3] In a graphical representation, $(a, k, P_k) \in OR$ means that k-th output of activity a is linked to place P_k. P_k is referred to as an output place of a.

C. Hierarchical OSAN Models

OSAN model as in Definition 3 are a flat network of elements. Now we define *hierarchical object stochastic activity networks* (HOSANs) as a hierarchical extension of OSANs. HOSAN models provide facilities for composing a hierarchy of OSAN submodels using a new element called *super activity* (SA).

A super activity is a submodel of OSANs, which is composed of OSAN elements and other super activities. *Place fusion* provides a mechanism for interfacing super activities to other parts of an HOSAN model. A super activity may have some *input* and *output fusion places* that are a subset of the simple or coloured places of OSANs. Fusion places will be bound to the actual places of the container HOSAN model or super activity.

Super activities are defined as *super activity classes.* Each super activity class can be *instantiated* to be used as an element of the container HOSAN model or super activity.

HOSAN model is defined as follows:

Definition 6. *Hierarchical object stochastic activity network* (HOSAN) is defined as a 4-tuple $HOSAN = (OSAN, \delta, SA, BF)$ where:

- $OSAN$ is defined as in Definition 3,
- δ is a finite set of *super activity classes*, which will be defined later.
- SA is a set of *super activity instantiations*, which is defined as follows:

$$SA = \{(sa, sac) | \; sa \text{ is an identifier for a super activity instantiation and } sac \in \delta\}.$$

- BF is a *binding function* that binds each input or output fusion place, fp, of super activity, sa, to one of the places of the model ($p \in SP \cup CP$) and is defined as follows:

$$BF: FP_{SA} \rightarrow SP \cup CP,$$

where SP is the list of simple places, CP is the list of coloured places, SA is the list of super activities and FP_{SA} is defined as follows:

$$FP_{SA} = \bigcup_{sa \in SA} (IFP_{sa} \cup OFP_{sa})$$

To bind a fusion place, fp, to a place, p, both of them must be of the same type (i.e. if fp is a simple place, p must be a simple place too. Also, if fp and p are coloured, their token classes must be same: $TC(fp) = TC(p)$).

Now, we can define HOSAN super activity as follows:

Definition 7. An HOSAN *super activity class* (SAC) is defined as a 6-tuple $SAC = (ID, F, M, HOSAN, IFP, OFP)$ where:

- ID is the identifier of the *super activity class*.

- F is a finite set of *data fields.*
- M is a finite set of *methods.*
- *HOSAN* is defined as in Definition 5.
- *IFP* is a finite set of *input fusion places*, such that:

$$IFP \subseteq SP \cup CP,$$

- *OFP* is a finite set of *output fusion places*, such that:

$$OFP \subseteq SP \cup CP,\ IFP \cap OFP = \varnothing, \text{ and } IFP \cup OFP \neq \varnothing.$$

The above definition of super activity is a recursive definition. Therefore, we need to define the base case:

Definition 8. An HOSAN with $SA=\varnothing$ is equivalent to a flat OSAN model. Such a model is composed of no super activity.

The graphical representation of super activity class is shown in Fig. 4.

Figure 4. The graphical representation of super activity

D. An Example of OSAN Models

As an example of OSANs and HOSANs, we model a queuing network of six tandem and parallel queues.. The resulting model is appeared in Fig. 5. In Fig. 5(a), we have defined a token class called *TCustomer*, which defined two fields *ID* of type char with length 2 and *Pri* of type byte and a method *CheckCustomer*. Then, we have defined an OSAN super activity called *QSA* in Fig. 5(b). *QSA* models an *M/M/1/N* queue, for customers of class *TCustomer*. We have defined the input and output fusion places (*input* and *output*) of *QSA* and a coloured place called *QPlace* of class *TCustomer* and with *FIFO* selection policy. Then, we have constructed, as shown in Fig. 5(c), a queueing network of six parallel and tandem queues of type *QSA*.

E. Evaluation of OSAN Models

OSAN models can be evaluated by state space analysis techniques or discrete-event simulation. Methods for the evaluation of OSANs are quite similar to CSAN models. Details are presented in our other publications ([2] or [6]).

Analytic solution by deriving embedded continuous-time Markov chains (CTMCs) could not be always used even for ordinary SAN models. Since, if an SAN model includes a non-exponential timed activity or if its state-space is infinite, it may not be solved analytically by transformation of state space into CTMCs. In such cases, discrete-event simulation may be employed to solve the model. Therefore, simulation is a general technique for the evaluation of OSAN models.

Figure 5. HOSAN model of a queuing network: (a) *TCustomer* token class, (b) *QSA* super activity, (c) HOSAN model

V. CONCLUSIONS

In this paper, we have presented the informal and formal definitions, graphical notations, and an example of object stochastic activity networks. OSANs are an object-oriented extension of SAN Models. Elements of OSANs and their submodels, called super activity, are defined as classes.

The object-orientation of OSAN models helps them to be more useful for modeling and evaluation of software systems and business processes. As a case study and to demonstrate the application of OSANs on software systems, we have transformed the UML diagrams into analyzable OSAN models [18]. The derived models have been evaluated using *SANBuilder* modeling tool [5]. The results of this case study showed that OSANs can be used as a complement of UML on modeling and evaluation of software systems. The object-orientation of OSANs and the flexibility of having functions for activities are the key benefits of OSANs over other extension of Petri nets for their applications in this area.

To continue the current research, we have planned to work on the following topics in future:

- Introduce efficient techniques for the reachability analysis and analytic solution of OSANs,
- Introduce fast methods for simulation of OSANs, and
- Do more case studies, which use OSANs for modeling and analysis of software systems.

REFERENCES

[1] M. Abdollahi Azgomi and A. Movaghar, "Towards an object-oriented extension for stochastic activity networks," Proc. of 10th Workshop on Algorithms and Tools for Petri Nets (AWPN'03), Eichstätt, Germany, 2003, pp. 144-155.

[2] M. Abdollahi Azgomi and A. Movaghar, "Coloured stochastic activity networks: definitions and behaviour," Proc. of the 20th Annual UK Performance Eng. Workshop (UKPEW'04), Bradford, UK, July 7-8, 2004, pp. 297-308.

[3] M. Abdollahi Azgomi and A. Movaghar, "Modeling and evaluation with object stochastic activity networks," Proc. of 1st Int'l Conf. on Quantitative Evaluation of Systems (QEST'04), Enschede, The Netherlands, Sept. 27-30, 2004, pp. 326-327.

[4] M. Abdollahi Azgomi and A. Movaghar, "Hierarchical stochastic activity networks: formal definitions and behaviour," Int'l J. of Simulation, Sys., Sci. and Tech., vol. 6, 2005, pp. 56-66.

[5] M. Abdollahi Azgomi and A. Movaghar, "A modeling tool for hierarchical stochastic activity networks," Simulation Modelling Practice and Theory, vol. 13, no. 6, Elsevier Science B.V., 2005, pp. 505-524.

[6] M. Abdollahi Azgomi, High-level extensions for stochastic activity networks: theories, tools and applications, Ph.D. dissertation, Dept. Comp. Eng., Sharif Univ. of Tech., Tehran, Iran, 2005. (*in Persian*)

[7] M. Ajmone Marsan, G. Balbo and G. Conte, "A class of generalized stochastic Petri nets for performance evaluation of multiprocessors systems," ACM Trans. Comp. Sys., vol. 2, no. 2, 1984, pp. 93-122.

[8] M. Baldassari, and G. Bruno, "An environment for object-oriented conceptual programming based on PROT nets," Advances in Petri Nets 1988, LNCS, vol. 340, Springer, 1988, pp. 1-19.

[9] R. Bastide and P. Palanque, "Cooperative objects: a concurrent, Petri net based, object-oriented language," IEEE System, Man and Cybernetics, 1993, pp. 286-291.

[10] R. Bastide, "Approaches in unifying Petri nets and the object-oriented approach," Proc. of 1st Workshop on OOP and Models of Concurrency (OO-MC'95), ICATPN'95, 1995.

[11] E. Battiston, F. de Cindio and G. Mauri, "OBJSA nets: a class of high-level nets having objects as domains," Proc. of APN'88, LNCS, vol. 340, Springer, 1988, pp. 20-43.

[12] E. Battiston, A. Chizzoni and F. de Cindio, "Inheritance and concurrency in CLOWN," Proc. of 1st Workshop on OOP and Models of Concurrency (OO-MC'95), ICATPN'95, 1995.

[13] D. Buchs, and N. Guelfi, "A concurrent object oriented Petri nets approach for system specification," Proc. of ICATPN'91, LNCS, Springer, 1991, pp. 432- 454.

[14] D.D. Deavours and et al., "The Möbius framework and its implementation," IEEE Trans. on Soft. Eng., vol. 28, no. 10, 2002, pp. 956-969.

[15] N. Guelfi et al., "Comparison of object-oriented formal methods," Tech. Rep. of the ESPIRIT Proj. #20072, Dept. of CS, Univ. of Geneva, 1997.

[16] J.E. Hong and D.-H. Bae, "Software modeling and analysis using hierarchical object-oriented Petri nets," J. of Info. Sci., vol. 130, 2000, pp. 133-164.

[17] K. Jensen, "An introduction to the theoretical aspects of colored Petri nets," in de Bakker, J.W., de Roever, W.-P. and Rozenberg, G. (eds.), A Decade of Concurrency, LNCS, vol. 803, Springer, 1994, pp. 230-272.

[18] A. Kamandi, M. Abdollahi Azgomi, and A. Movaghar. "Transformation of UML models into analyzable OSAN models," Elect. Notes in Theo. Comput. Sci. (ENTCS), vol. 159, Elsevier Science B.V., 2006, pp. 3-22.

[19] O. Kummer and F. Wienberg, "Renew - the reference net workshop," Petri Net Newsletter, vol. 56 , 1999, pp. 12-16.

[20] C. Maier and D. Moltz, "Object coloured Petri nets - a formal technique for object-oriented modeling, in Agha, G. et al. (eds.): Concurrent OOP and Petri nets, LNCS, vol. 2001, Springer, 2001, pp. 406-427.

[21] M. K. Molloy, "Performance analysis using stochastic Petri nets," IEEE Trans. on Comp., vol. C-31, 1982, pp. 913-917.

[22] C.A. Lakos and C.D. Keen, "LOOPN-language for object-oriented Petri nets," Proc. of SCS Multiconf. on O.O. Sim., 1991, pp. 22-30.

[23] C.A. Lakos and C.D. Keen, "LOOPN++-a new language for object-oriented Petri nets," Proc. of ESM'94, 1994, pp. 369-374.

[24] C.A. Lakos, "From coloured Petri nets to object Petri nets," Proc. of ICATPN'95, LNCS, vol. 935, Springer, 1995, pp. 278-297.

[25] A. Movaghar. and J.F. Meyer, "Performability modeling with stochastic activity networks," Proc. of the 1984 Real-Time Systems Symp., Austin, TX, USA, pp. 1984, pp. 215-224.

[26] A. Movaghar, "Stochastic activity networks: a new definition and some properties," Scientia Iranica, vol. 8, no. 4, 2001, pp. 303-311.

[27] C. Sibertin-Blanc, "Cooperative nets," Proc. of ICATPN'94, LNCS, Springer, 1994.

[28] W.H. Sanders et al: "The UltraSAN modeling environment," Performance Evaluation, Elsevier, vol. 24, 1995, pp. 1-33.

[29] R. Valk, "Petri nets as token objects," An introduction to elementary object nets," Proc. of ICTAPN'98, LNCS, vol. 1420, Springer, 1998, pp. 1-25.

[30] K.M. van Hee and P.A.C. Verkoulen, "Integration of a data model and high-level Petri nets," Proc. of ICATPN'91, LNCS, Springer, 1991.

[31] P.A.C. Verkoulen, "Integrated information systems design: an approach based on object-oriented concepts and Petri nets, Ph.D. Thesis, Technical Univ. of Eindhoven, the Netherlands, 1993.

[32] T. Vojnar, "Towards formal analysis and verification over state spaces of object-oriented Petri nets, Doctoral Thesis, Brno Univ. of Techn., 2001.