Modeling and Evaluation of Software Systems with Object Stochastic Activity Networks

M. Abdollahi Azgomi, A. Kamandi and A. Movaghar

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 have several distinguishing properties, which are not found in other object-oriented or high-level extensions of Petri nets. In this paper, we will present the definitions, behavior and methods for the solution of OSAN models. Then, we will present the result of a case study on transformation of the UML use case and activity diagrams into OSAN models. The derived OSAN models have been evaluated using our modeling tool called SANBuilder. 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.


1. Introduction

Stochastic activity networks (SANs) [Movaghar84] are a stochastic generalization of Petri nets (PNs) [Peterson81]. These models are more powerful and flexible than most other stochastic extensions of Petri nets including notable models such as stochastic Petri nets (SPNs) [Molloy82] and generalized stochastic Petri nets (GSPNs) [Ajmone86].

SAN models have been used to evaluate the performance, dependability and performability of a wide range of systems. These models have been used as a modeling formalism in several modeling tools, such as UltraSAN [Sanders95] and Möbius [Deavours02].

In order to integrate the concepts of object-orientation (OO) into SANs and to overcome some restrictions of these models, we have recently introduced a new extension for SANs, called object stochastic activity networks (OSANs): OSANs = SANs + Hierarchy + Colour + OO [Abdollahi03c, Abdollahi04d].

Primitives of OSANs and their submodels (called super activity) are defined as classes. Tokens of OSANs are objects of a user-defined colour class. These models adapt some useful ideas and features from the existing high-level and object-oriented extensions of Petri nets. However, OSANs have several distinguishing properties that are not found in the existing OO or high-level extensions of Petri nets.

We have implemented OSAN models in SANBuilder modeling tool [Abdollahi04a]. SANBuilder provides features for construction, animation, simulation and analytic solution of OSAN and other SAN-based models.

As a case study, we have transformed the unified modeling language (UML) use case and activity diagrams into OSAN models. The derived OSAN models have been evaluated using SANBuilder modeling tool. The results of this case study proved that OSANs can be used as a complement or even an alternative of the UML for modeling and evaluation of software systems.

This paper is organized as follows. Sec. 2 mentions the motivations of this work and gives a survey on related work. In Sec. 3, the informal definitions, graphical notations, Meta model and an example of OSAN models are presented. The behavior of OSANs is discussed in Sec. 4. In Sec. 5, methods for the solution of OSAN models are introduced. Sec. 6 gives an overview of a modeling tool for OSANs. The transformation of UML use case and activity diagrams into OSANs and their evaluation is presented in Sec. 7. A summary of advantages of OSANs for application in software systems is mentioned in Sec. 8. Finally, in Sec. 9, some concluding remarks are mentioned.

2. Motivations

In this section, some motivations of this work and a survey on related work on integration of object-orientation and Petri nets are presented.
2.1. Restrictions of SAN Models

SAN models have the following restrictions:

1. **SANs are flat models.** Originally, SANs are defined as a flat model. The Replicate/Join construct [Sanders91] has been proposed for constructing hierarchical and composed SAN models with reduced state spaces. However, there is a set of issues with this construct. It is limited to a tree-like structure. This construct has been extended and generalized in the work of [Stillman99] on the graph composition formalism in the Möbius modeling framework [Deavous01]. Using these methods, a model is constructed in a bottom-up manner using Join and Replicate operations [Sanders91]. While, top-down model construction, especially, for large models, is more appropriate. On the other hand, the use of these composition formalisms is specific to the UltraSAN and Möbius modeling tools and not SANs in general. These composition formalisms have not been defined formally along with the definition of SANs. However, these techniques are not appropriate for modeling software systems, because they cannot naturally capture the hierarchy inherent into a hierarchical software system. Therefore, a technique that encapsulates hierarchies is required. **Automatic code generation** from hierarchical models of a software system is another requirement that should be considered in a hierarchical extension of SANs.

2. **SANs lack facilities for incremental modeling.** There is a lack of facilities for constructing complex models incrementally, by starting with abstract components and easily replacing them with detailed and enhanced components. Here, again a top-down paradigm for model construction is needed.

3. **SANs lack facilities for reusability.** Since SAN models are flat, using a part of an existing model, as a component for constructing a new one is difficult. Specially, making and sharing a global repository of submodels with well-defined interfaces is not easily possible.

4. **SANs lack facilities for data manipulation.** Tokens of SANs are simple and of integer type. A language allows a person to put more meaning into constructs, such as variable types. For example, types make expressing things conceptual, rather than forcing the user to translate their concept into non-negative integers.

5. **SANs are not object-oriented models.** Object-orientation (OO) 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 [Hong90]. OO concepts allow constructing models easily, intuitively and naturally.

2.2. Related Work

The absence of structuring capabilities has been one of the main criticisms raised against high-level Petri nets [Gueff97]. 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 follows [Bastide95]:

1. **Objects inside Petri nets**: Purpose of this trend is to increase in various ways the amount of information that is held in tokens. The tokens that flow in the nets are instances of object classes, described in some kinds of OO languages. The occurrence of a transition may create or destroy token/objects, or only move them from one place to another.

2. **Petri nets inside objects**: In this approach, nets are used to model the inner behavior of objects and the current marking of the net models the inner state of an object and transitions in the net may be used to model the execution of a method by this object.

C.A. Lakos has categorized **object-oriented Petri nets** (OOPNs) as follows [Lakos95]:

- **Object-based Petri net (OBPN)**: OBPN enhances the modular coloured Petri nets (MCPNs) with multiple levels of activity, by allowing tokens to be object identifiers of pages. The name of this formalism has been derived from the classification of programming languages into object-based and object-oriented. Object-based languages have encapsulation and instantiation of classes, but do not have inheritance nor the associated polymorphism and dynamic binding.

- **Object-oriented Petri net (OOPN)**: OOPN is derived from object-based Petri nets by including the notion of inheritance together with the associated polymorphism and dynamic binding. In OOPNs, pages are referred to as classes, which can be related by inheritance.

- **Object Petri nets (OPN)**: OPN proposes a complete integration of object-oriented concepts, such as inheritance, polymorphism and dynamic binding into Petri nets.
Different styles and dialects of OOPNs are described in the literature. A (partial) historical list of OOPNs is shown in Tab. 1. For a detailed comparison, please see [Bastide95, Lakos95, Guelfi97, Hong00].

<table>
<thead>
<tr>
<th>Year</th>
<th>Name</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>1984</td>
<td>PROT nets</td>
<td>[Baldasarri88]</td>
</tr>
<tr>
<td>1988</td>
<td>OBJISA nets</td>
<td>[Battiston88]</td>
</tr>
<tr>
<td>1991</td>
<td>SimCon object model</td>
<td>[vaniHee91]</td>
</tr>
<tr>
<td>1991</td>
<td>Language for O-O Petri net (LOOPEN)</td>
<td>[Lakos91]</td>
</tr>
<tr>
<td>1991</td>
<td>Concurrent O-O Petri net (CO-OPN)</td>
<td>[Bucio91]</td>
</tr>
<tr>
<td>1993</td>
<td>Expressive comfort</td>
<td>[Verkoulen93]</td>
</tr>
<tr>
<td>1993</td>
<td>Cooperative objects</td>
<td>[Bastide93]</td>
</tr>
<tr>
<td>1994</td>
<td>Cooperative nets</td>
<td>[Sibertin94]</td>
</tr>
<tr>
<td>1994</td>
<td>LOOPEN++</td>
<td>[Lakos94]</td>
</tr>
<tr>
<td>1995</td>
<td>Object Petri nets (OPNs)</td>
<td>[Lakos95]</td>
</tr>
<tr>
<td>1995</td>
<td>Class oriented with nets (CROWN)</td>
<td>[Battistor95]</td>
</tr>
<tr>
<td>1997</td>
<td>PNTalk (PN + Smalltalk)</td>
<td>[Vojnar01]</td>
</tr>
<tr>
<td>1998</td>
<td>Object nets</td>
<td>[Valk98]</td>
</tr>
<tr>
<td>1999</td>
<td>Reference nets</td>
<td>[Kummer99]</td>
</tr>
<tr>
<td>2000</td>
<td>Hierarchical O-O Nets (HOONets)</td>
<td>[Hong00]</td>
</tr>
<tr>
<td>2001</td>
<td>Object coloured Petri nets (OCP nets)</td>
<td>[Mace01]</td>
</tr>
</tbody>
</table>

3. Object Stochastic Activity Networks

In this section, we introduce object stochastic activity networks (OSANs), including its elements, graphical notations, Meta model and an example. The definitions presented in this section are the finalized version of that appeared in [Abdollahi03c].

Our base model for the definition of OSAN models is a new definition of SANs [Movaghar01] that is slightly different from their original definition [Movaghar84, Movaghar85, Sanders01]. In the original definition of SANs, as appeared in 1984, a model must first be checked for well-behavedness. This check is in general undecidable and is computationally complex for most models. However, in a new definition of SANs as appeared in [Movaghar01], no such check will ever be necessary because the models are ensured to be well-behaved. For more information about the differences between these two definitions of SANs and their formal definitions and properties, please see [Movaghar01].

3.1. Elements of OSAN Models

The elements of OSANs are simple place, coloured place, instantaneous activity and timed activity. In OSANs, input and output gates of SANs no longer exist. These elements are encapsulated into instantaneous and timed activities. In addition to the place of SANs that hold values of integer type, OSANs have another kind of places that hold lists of tokens (objects) of a user-defined colour class. To distinguish these two kinds of places, we call the former simple place and the later coloured place.

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

3.1.1. Simple place. A simple place in OSANs is similar to SANs and Petri nets. A simple place has a unique identifier and is equivalent to a counter of integer type in programming languages. Graphically, a simple place is drawn as a circle ( ). A simple place may hold zero or more tokens, which describe its marking. A token is drawn by a black dot inside the circle of a place (e.g. ). The number of tokens inside a simple place is used in the predicates and functions of timed or instantaneous activities.

Marking of simple place is defined as follows:

Definition 3.1  
Marking of a simple place is defined as a function: 

where SP denotes the set of simple places and N denotes the set of natural numbers.

3.1.2. Colour class. Coloured tokens that are hold by coloured places are objects of a user-defined colour class. A colour class has one or more fields and zero or more methods. Methods of a colour 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 can be determined by private or public keywords. The syntax for the colour class definition is quite similar to C++ or Java. The definition of colour class is as follows:

Definition 3.2  
Colour class is defined as a triplet (ID, F, M), where:
- ID is a unique identifier of the class,
- F is a set of fields, and
- M is a set of methods.

3.1.3. Coloured place. A coloured place holds a list of tokens of a specified colour class. Coloured tokens are removed from the list of tokens based on the specified scheduling strategy for the coloured place. The scheduling strategy can be selected from the set of {NONE, FIFO, LIFO, PRI}. If the scheduling strategy of a coloured place is NONE, the list of tokens is unordered; otherwise, it is an ordered list. 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\(^1\) and functions\(^2\) or can be read or manipulated by functions of activities. The following are some definitions related to coloured place:

**Definition 3.3**

A token list, denoted by \(l(cc)\), is a list of objects or tokens of colour class \(cc\). The following operations are defined on a token list:
- \(\text{Clear}()\): makes the list empty,
- \(\text{Add}(t)\): adds an object, \(t\), of type \(cc\) to the list,
- \(\text{Token}([i])\): provides access to the first \((i\text{-th}, \text{if } SS=\text{NONE})\) token of the list,
- \(\text{Remove}([i])\): removes and returns the first \((i\text{-th}, \text{if } SS=\text{NONE})\) token of the list,
- \(\text{Size}()\): gives the number of tokens inside the place,
- \(\text{SearchToken}(f, v)\): if \(SS=\text{NONE}\), will search in the token list for a token that the value for field \(f\) is \(v\),
- \(\text{Sort}(OF)\): reorders the tokens of the list, where \(OF\) is an order function that defines the order of each token in the list.

**Definition 3.4**

The set of all possible instances of a token list of type \(cc\) is denoted by \(L(cc)\).

For a coloured place, the number of coloured tokens inside the list and the elements of the list of coloured tokens, represent marking.

**Definition 3.5**

Marking of a coloured place is defined as a function \(\mu: CP \rightarrow L(CP)\), where \(CP\) denotes the set of coloured places and for each \(P_i \in CP\), \(\mu(P_i) \in L(P_i)\).

Colored place is defined as follows:

**Definition 3.6**

A coloured place is defined as a 5-tuple \((ID, CC, SS, TL, OF)\), where:
- \(ID\) is the identifier of the place,
- \(CC\) is the identifier a colour class.
- \(SS \in \{\text{NONE}, \text{FIFO}, \text{LIFO}, \text{PRI}\}\) is a scheduling strategy for the place.
- \(TL(cc)\) is the token list of the place.
- \(OF\) is the order function of the place. \(OF\) is a computable partial function, which determines the order of tokens inside the token list of the place. \(OF\) is a function of the state of \(n\) places and is defined as:

\[
OF: M_1 \times \cdots \times M_n \rightarrow R,
\]

where \(n \geq 0\) and \(M_i = N\) if \(P_i\) is a simple place and \(M_i = L(P_i)\) if \(P_i\) is a coloured place. If \(n \geq 1\), then the order function is state-dependent, otherwise it is state-independent. If \(SS \neq PRI\), then \(n = 0\) and the order function is an identity function \((OF \rightarrow 1)\).

\[z(CCName, SS)\]

**Fig. 1.** The graphical notation of a coloured place

The graphical representation of a coloured place is shown in Fig. 1, where \(CCName\) is the name of a previously defined colour class and \(SS\) is the scheduling strategy of the place.

The following are some clarifications of coloured places:

- Is it really needed to have two kinds of places (simple and coloured)? It is possible to use a coloured place instead of a simple place. However, since no token list is maintained for a simple place and its state is represented by an integer, it is more efficient and convenient to use a simple place, rather than a coloured place.
- What is the scheduling strategy good for? In many applications, it is needed to guarantee that tokens are removed from the input queue by a specified order. The queuing systems with different disciplines, such as FIFO, LIFO, or priority are used in different applications for these purposes. In queuing Petri nets (QPNs) [Bause98], such a combination of queuing systems with Petri nets is done in a completely different manner. Using the specified scheduling strategy for the tokens, modeling such kind of systems will be really easy.
- What does the marking of a coloured place represent? Regarding the specified scheduling strategy for a coloured place, its marking represents an ordered or unordered list of tokens. If a FIFO, LIFO or PRI scheduling strategy is specified, the marking of the coloured place will represent an ordered list of tokens. Otherwise, it will represent an unordered list tokens. It is obvious that the state space of OSANs will be very complex and large.
- What type of priorities (state-dependent or state-independent) can be modeled by a coloured place? Both types of priorities can be modeled by PRI scheduling strategy. For state-dependent priorities, an automatic reordering of the token list of the coloured place after a state change is needed. This is accomplished by an automatic execution of the Sort function of the token list. Therefore, a high degree of optimization is required by modeling tools to provide efficiency for state space generation or simulation of OSAN models.

\(^1\) Enabling predicate of activities or reactivation predicate of timed activities.

\(^2\) Activity functions, enabling rate functions of timed activities, case probability functions of instantaneous activities, or order function of coloured places.
3.1.4. 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. The reason for this integration is to avoid having a non-standard element that is not found in other standard extensions of Petri nets. A timed activity is connected directly to some input and output places. In order to interface with other elements of the model, a timed activity must have at least one input place or one output place. Input places of a timed activity are read-only, while the marking of output places can be changed by the activity function. The set of input and output places can intersect. The definition of timed activity is as follows:

**Definition 3.7**
A timed activity with \( m \) input places and \( n \) output places, is defined as a 6-tuple \((ID, e, F, \rho, II, G)\), where:
- \( ID \) is the identifier of the activity,
- \( e \) is the enabling predicate of the activity, which is defined as:
  \[
e: M_1 \times \ldots \times M_m \rightarrow \{true, false\},
\]
  where \( M_i = N \) if \( P_i \) is a simple place and \( M_i = L(P_i) \) if \( P_i \) is a coloured place,
- \( F \) is the activity time distribution function of the activity, which is defined as:
  \[
F = \{ F(\cdot|\mu) ; \mu \in M_1 \times \ldots \times M_m \},
\]
  where \( M_i \) is defined as before and for \( \mu \in M_1 \times \ldots \times M_m \), \( F(\cdot|\mu) \) is a probability distribution function,
- \( \rho \) is the enabling rate function of the activity, which is the activity execution speed and is defined as:
  \[
\rho: M_1 \times \ldots \times M_m \rightarrow R^+_+,
\]
  where \( M_i \) is defined as before and \( R^+_+ \) represents the set of non-negative real numbers,
- \( II \) is the reactivation predicate of the activity, which is defined as:
  \[
II: M_1 \times \ldots \times M_m \rightarrow \{true, false\},
\]
  where \( M_i \) is defined as before,
- \( G \) is the activity function, which is defined as:
  \[
G: M_1 \times \ldots \times M_m \rightarrow M_1 \times \ldots \times M_m,
\]
  where \( M_i \) is defined as before.

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

3.1.5. 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. A similar approach has also been used in [Ajmone87] for GSPNs. An instantaneous activity in OSANs is an encapsulation of instantaneous activity, input gate and output gate of the original SANs. The definition of instantaneous activity is as follows:

**Definition 3.8**
An instantaneous activity with \( m \) input places and \( n \) output places, is defined as a 4-tuple \((ID, e, C, G)\), where:
- \( ID \) is the identifier of the activity,
- \( e \) and \( G \) are defined as before, and
- \( C \) is the case probability function, which is defined as:
  \[
C: M_1 \times \ldots \times M_m \rightarrow [0, 1],
\]
  where \( M_i \) is defined as before.

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

3.1.6. Object instantiation. Instantiation is the definition of different objects or instance variables with unique identifiers of previously defined classes (simple place, coloured class, timed or instantaneous activity and global variables) in an OSAN model. The definition of instantiation is as follows:

**Definition 3.9**
Object instantiation is defined as a triplet \((CID, OID, IL)\), where:
- \( CID \) is the identifier of a class,
- \( OID \) is the unique identifier of an instance variable, and
- \( IL \) is a list of (initial) values of type OID.

3.1.7. Relation. A relation in OSANs is equivalent to an arc in Petri nets that connects two nodes. Two kinds
of relations are distinguished: input relation (connects a place to an activity) and output relation (connects an activity to a place). A relation is represented by an arc in the graphical representation of OSAN model.

### 3.2. Definition of Flat OSAN Models

An object stochastic activity network (OSAN) is the compound class of an OSAN model that describes how various objects of a model interact together. OSAN is defined as follows:

**Definition 3.10**

Object stochastic activity network (OSAN) is defined as a 7-tuple \( (\Sigma, SP, CP, IA, TA, IR, OR) \), where:

- \( \Sigma \) is a set of colour classes,
- \( SP \) is a set of simple places,
- \( CP \) is a set of coloured places,
- \( TA \) is a set of timed activities,
- \( IA \) is a set of instantaneous activities,
- \( IR \subseteq (SP \cup CP) \times \{1, \ldots, [SP \cup CP]\} \times (IA \cup TA) \) is the input relation. \( IR \) satisfies the following conditions:
  - For any \( (P_1, i, a) \in IR \) such that \( a \) has \( k \) inputs, \( i \leq k \).
  - For any \( a \in IA \cup TA \) with \( k \) inputs and \( i \in N \), \( i \leq k \), there exist \( P_1 \in (SP \cup CP) \) such that \( (P_1, i, a) \in IR \), and
  - For any \( (P_1, i, a_1), (P_1, j, a_2) \in IR \), \( i = j \) and \( a_1 = a_2 \).

In a graphical representation, \((P_k, k, a) \in IR\) means that place \( P_k \) is linked to \( k \)-th input of activity \( a \).

- \( OR \subseteq (IA \cup TA) \times \{1, \ldots, [SP \cup CP]\} \times (SP \cup CP) \) is the output relation. \( OR \) satisfies the following conditions:
  - For any \( (a, i, P_1) \in OR \) such that \( a \) has \( k \) outputs, \( i \leq n \).
  - 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 \( a_1 = a_2 \).

In a graphical representation, \((a, k, P_k) \in OR\) means that \( k \)-th output of activity \( a \) is linked to place \( P_k \).

### 3.3. An Example of A Flat OSAN Model

To illustrate some properties of OSAN models, we present a simple model of a closed queueing system for a computer system with three tasks. Each task has three properties: an identity, a priority (1 or 2) and a processor type. The graphical representation of the model with an initial marking is shown in Fig. 4(a). The model is composed of three coloured places \((SysBuf, PBuf1, \text{and} PBuf2)\) and three timed activities \((Route, Proc1, \text{and} Proc2)\). The scheduling strategy for \(SysBuf\) is PRI and for \(PBuf1\) and \(PBuf2\) is FIFO. The order function of \(SysBuf\), which is defined in the activity table (Tab. 2), determines the priority of tasks inside system buffer.

This model is constructed by \textsc{SANBuilder} modeling tool, which will be introduced in Sec. 6 of this paper. In the initial marking of the model as appeared in Fig. 4(a), there are three coloured tokens inside \(SysBuf\), which are distinguished by their colours (brown, green and magenta). The token \(\{ID=a', Pri=1, ProcType=1\}\) is shown in brown colour, the token \(\{ID=b', Pri=1, ProcType=2\}\) in green and the token \(\{ID=c', Pri=2, ProcType=2\}\) in magenta. The colour association by \textsc{SANBuilder} to the tokens is quite random. Two other places \((PBuf1, \text{and} PBuf2)\) are empty.

Each OSAN model has a class table for the definition of token classes and an activity table for the definition of timed and instantaneous activities of the model. The class and activity tables, corresponding to the model of Fig. 4, are displayed in Tab. 2 and 3, respectively. In Tab. 2, we have defined a token class, named \textsc{TTask}. \textsc{TTask} has three fields \((ID, Pri, \text{and} ProcType)\) and a method, named CheckTask that does some checking on tasks. In Tab. 3, we have also defined three timed activities \((Route, Proc1, \text{and} Proc2)\).

Marking of the net after several firing of the activities of the model is depicted in Fig. 4(b). For this purpose, we have employed the animation feature of \textsc{SANBuilder}. This feature executes the model step-by-step and animates the result of the model execution on the graphical user interface.

### 3.4. Hierarchical OSAN Models

OSAN models as in Definition 3.10 are flat networks 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 by introducing a new element called super activity (SA).
Tab. 2. Class table of the OSAN model of Fig. 6

<table>
<thead>
<tr>
<th>Class Name</th>
<th>Fields/Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>TTask</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Char ID;</td>
</tr>
<tr>
<td></td>
<td>byte Pri;</td>
</tr>
<tr>
<td></td>
<td>Fields</td>
</tr>
<tr>
<td></td>
<td>byte ProcType;</td>
</tr>
<tr>
<td></td>
<td>TTask</td>
</tr>
<tr>
<td></td>
<td>Methods</td>
</tr>
</tbody>
</table>
|            | bool CheckTask()
|            | { //Do some checking } |
| SysBuf (TTask) | OrderFunction return Token->Pri; |

Tab. 3. Activity table of the OSAN model of Fig. 6

<table>
<thead>
<tr>
<th>Activity</th>
<th>Predicate/Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Route</td>
<td></td>
</tr>
<tr>
<td></td>
<td>TTask-&gt;mpTask;</td>
</tr>
<tr>
<td></td>
<td>TTask = SysBuf-&gt;Remove();</td>
</tr>
<tr>
<td></td>
<td>TTask-&gt;CheckTask();</td>
</tr>
<tr>
<td></td>
<td>if (TTask-&gt;ProcType == 1)</td>
</tr>
<tr>
<td></td>
<td>PhBuf1-&gt;Add(TTask);</td>
</tr>
<tr>
<td></td>
<td>else if (TTask-&gt;ProcType == 2)</td>
</tr>
<tr>
<td></td>
<td>PhBuf2-&gt;Add(TTask);</td>
</tr>
<tr>
<td>Proc1</td>
<td>EnablingPredicate PhBuf1-&gt;Start() &gt; 0</td>
</tr>
<tr>
<td></td>
<td>ActivityFunction</td>
</tr>
<tr>
<td></td>
<td>SysBuf-&gt;Add(PhBuf1)-&gt;Remove1();</td>
</tr>
<tr>
<td>Proc2</td>
<td>EnablingPredicate PhBuf2-&gt;Start() &gt; 0</td>
</tr>
<tr>
<td></td>
<td>ActivityFunction</td>
</tr>
<tr>
<td></td>
<td>SysBuf-&gt;Add(PhBuf2)-&gt;Remove1();</td>
</tr>
<tr>
<td></td>
<td>OutputFunction</td>
</tr>
<tr>
<td></td>
<td>SysBuf-&gt;Add(PhBuf1)-&gt;Remove1();</td>
</tr>
</tbody>
</table>

A super activity is a submodel of OSANs, which is composed of a finite number 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 zero or more input and output fusion places that are a subset of simple and coloured places of OSANs. Fusion places will be bound to the places of the composer HOSAN model or super activity.

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

HOSAN super activity class is defined as follows:

**Definition 3.11**

An HOSAN super activity class (SAC) is defined as a 5-tuple SAC = (ID, OSAN, IFP, OFP, INIT) where:

- ID is the identifier of super activity,
- OSAN is defined as in Definition 3.10,
- IFP is a finite set of input fusion places, such that:
  - IFP ⊆ SP ∪ CP,
- OFP is a finite set of output fusion places, such that:
  - OFP ⊆ SP ∪ CP,
  - IFP ∩ OFP = ∅,
  - IFP ∪ OFP ≠ ∅
- INIT is the initialization method of the super activity, which initializes the instance of the super activity.

In a graphical representation, fusion places are depicted as (for simple places) or (for coloured places). The graphical representation of an SA with m inputs and n outputs is shown in Fig. 4.

HOSAN model is defined as follows:

**Definition 3.12**

Hierarchical object stochastic activity network (HOSAN) is defined as a 4-tuple HOSAN = (OSAN, δ, SA, FF) where:

- OSAN is defined as in Definition 3.10,
- δ is a finite set of super activity classes,
- SA is a finite set of super activities. To each sa ∈ SA is associated a super activity class sac ∈ δ,
- FF is a fusion function that is defined as follows: FF: SA × FP → SP ∪ CP.

The fusion function, FF, maps each fusion place, fp, of sa (where sa ∈ SA, fp ∈ sa.FP and sa.FP = sa.IFP ∪ sa.OFP) to a place p ∈ SP ∪ CP (where fp and p must be of the same type - i.e. if fp is a simple place, p must be a simple place and if fp is a coloured place, p must be a coloured place).
3.5. Well-Defined HOSAN Models

HOSANs are hierarchical models composed of simple or coloured places and timed, instantaneous or super activities. A super activity itself is composed of some simple or coloured places and timed, instantaneous or other super activities. Therefore, it is needed to check for a cycle in the nested usage of super activities. This check is done on the composition graph of the model. In this graph, each node is a super activity and a composition relation between two super activities determines an arc. The root node of the composition graph is the HOSAN model. If there is no cycle in the composition graph of an HOSAN model, it is well-defined and can be analyzed by a reachability analysis or discrete-event simulation method.

Definition 3.13
Composition graph (CG) of an HOSAN model is defined as a triplet \( DG = (N, A, R) \), where:
- \( N = \{\text{HOSAN}\} \cup \text{SA} \) is the set of nodes of the graph, such that:
  - \( \text{HOSAN} \) is the root node of the model, and
  - \( \text{SA} \) is the set of super activities of the model,
- \( A \) is the set of arcs of the graph, and
- \( R \) is the composition relation between two nodes such that:
  \[ R: A \rightarrow N \times N. \]

Definition 3.14
An HOSAN model is well-defined, if its composition graph is finite and acyclic.

3.6. OSAN Meta Model

The Meta model of OSANs is shown in Fig. 5.

![Fig. 5. OSAN Meta model](image)

The Meta model of OSANs has the following elements:

- \( N: \text{Node} \{\text{ID}\} \)
- \( R: \text{Relation} \{\text{NodeID, NodeID}\} \)
- \( P: \text{Place} \rightarrow \text{Node} \{\text{ID, State, Init(), Add(), Remove(), Clear()}\} \)
- \( A: \text{Activity} \rightarrow \text{Node} \{\text{ID, Place IP[], Place OP[]}, \text{EnablingPredicate()}, \text{InputFunction()}, \text{OutputFunction()}\} \)
- \( \text{SP}: \text{SimplePlace} \rightarrow \text{Place} \{\text{ID, Integer State, Init(n), Add(n), Remove(n), Clear()}, \text{Mark()}\} \)
- \( \text{CC}: \text{ColourClass} \{\text{Field1, .., Fieldm, Method1(), …, Methodn()}\} \)
- \( \text{CP}: \text{ColouredPlace} \rightarrow \text{Place} \{\text{ID, ColourClass State[, SS, Init(ss, il), OrderFunction()}, \text{Add(i), Remove(i]}, \text{Token(i)}, \text{Clear()}, \text{Size()}, \text{SearchToken()}, \text{Sort()}\} \)
- \( \text{TA}: \text{TimedActivity} \rightarrow \text{Activity} \{\text{ID, Place IP[], Place OP[]}, \text{EnablingPredicate()}, \text{ActivityDistribution()}, \text{EnablingRate()}, \text{RactivationPredicate()}, \text{ActivityFunction()}\} \)
- \( \text{IA}: \text{InstActivity} \rightarrow \text{Activity} \{\text{ID, Place IP[], Place OP[]}, \text{EnablingPredicate()}, \text{CaseProbability()}, \text{ActivityFunction()}\} \)
- \( \text{IR}: \text{InputRelation} \rightarrow \text{Relation} \{\text{Place, Activity}\} \)
- \( \text{OR}: \text{OutputRelation} \rightarrow \text{Relation} \{\text{Activity, Place}\} \)
- \( \text{SA}: \text{SuperActivity} \rightarrow \text{Activity} \{\text{ID, Place IP[], Place OP[]}, \text{EnablingPredicate=}false, \text{INIT()} = \{\}\} \)
- \( \text{OSAN}: \text{ObjectSAN} \{\text{Place P[], Activity A[], Relation R[]}, \text{INIT()}, \text{Field1, .., Fieldm, Method1(), …, Methodn()}\} \)

In the above list, those classes, fields or methods that are in italic are abstract and "\( \rightarrow \)" is used to show the specialization relation.

3.7. The Object-Orientation of OSANs

Among important characteristics of object-orientation, OSANs support encapsulation, polymorphism and inheritance. These characteristics are supported for colour classes and for super activities of OSANs.

The syntax and semantics of OO languages, like C++ and Java is used as inscription language for enabling predicate of activities or reactivation predicate of timed activities and activity functions, enabling rate functions of timed activities, case probability functions of instantaneous activities, or order function of coloured places.

The selection and usage of an OO language as inscription language is a tool specific feature. Based on the language selected, it will be possible to automatically generate source codes from OSAN models by a modeling tool.

4. Behavior of OSAN Models
An OSAN with a marking is a dynamic system. Here, we first define the marking of an OSAN model:

**Definition 3.15**
Consider an OSAN as in Definition 3.10. Marking of an OSAN is defined as an n-tuple μ = (μ1, ..., μn), where μi = μ(Pi), i = 1, ..., n and P1 ∈ SP ∪ CP. For example, the marking of the OSAN model in the initial marking of Fig. 4(a) is ([{c’, 2, 2} + {b’, 1, 2} + {a’, 1, 1}]), ([1], []). However, the marking displayed in Fig. 4(b) is ([], [{a’, 1, 1}], [{c’, 2, 2} + {b’, 1, 2}]).

An activity is enabled in a marking if its enabling predicate is true in that marking. More formally, we have:

**Definition 3.16**
Consider an OSAN as in Definition 3.10. A timed or instantaneous activity, a ∈ (IA ∪ TA), with m inputs, is enabled in a marking μ if its enabling predicate,

a.e(μ1, ..., μm) = true,

where μk = μ(Pk), for some Pk ∈ SP ∪ CP such that (Pki, i) ∈ IR, k = 1, ..., m. An activity is disabled in a marking if it is not enabled in that marking. A marking is stable if no instantaneous activity is enabled in that marking. A marking is unstable if it is not stable.

In an OSAN model, a marking change only if an activity completes. In a stable marking, only one of the enabled timed activities is allowed to complete. When there is more than one enabled timed activity, the choice of which activity to complete first is done stochastically. In an unstable marking, only one of the enabled instantaneous activities may complete (i.e., enabled instantaneous activities have priority over enabled timed activities for completion). When there is more than one enabled instantaneous activity, the choice of which activity to complete first is made probabilistically. More specifically, let M be an OSAN as in Definition 3.10. Suppose, M is in an unstable marking μ. Let A’ be the set of enabled instantaneous activities of M in μ. Then, a ∈ A’ completes with probability 1/n, where

\[ a.C(\mu) = \frac{a.C(\mu)}{\sum_{a \in A'} a.C(\mu)} \tag{3.1} \]

When instantaneous activities are enabled they complete instantaneously. Enabled timed activities, on the other hand, require some time to complete. A timed activity becomes active as soon as it is enabled and remains so until it completes; otherwise, it is inactive.

Consider an OSAN M as in Definition 3.10. Suppose, at time t, a timed activity completes, and μ is the stable marking of M immediately after t. A timed activity, a, is activated at t, if a is enabled in μ and one of the following occurs:

- a is inactive immediately before t,
- a completes at t,
- a.H(μ) = TRUE.

Whenever the above happens, a is assigned an activity time, τ, where τ is a random variable with probability distribution function a.F(τ(μ)). When a timed activity a is enabled in a stable marking μ, it is processed with a rate a.P(μ). A timed activity completes whenever it is processed for its activity time. Upon completion of an activity, the next marking occurs immediately.

When an activity completes, it may change the marking of its input and output places. This change is governed by the activity function and is done in two steps as follows. First, the marking of its input and output places may change due to the activity functions, resulting in an intermediate marking. Second, an additional step is required if there are some coloured places with PRI scheduling strategy in the net. Since, based on the definition, the priority of tokens inside coloured places can be dynamic and state-dependent, a reordering of the tokens inside such places is required. This reordering will consequently change the marking.

More specifically, let us consider an OSAN as in Definition 3.10. Suppose an activity, a, completes in a marking μ. The next marking μ’ is determined in two steps as follows. First, an intermediary marking μ’ is obtained from μ by the activity function of a. μ’ is then determined from μ” by reordering the tokens of coloured places with PRI scheduling strategy.

More formally, μ’ and μ” are defined as follows:

- **Step 1:** For any Pk ∈ SP ∪ CP:

  1.1. If P1 is not an input or output place of a,
  \[ \mu’(P1) = \mu(\mu) \]

  1.2. If a has m inputs and n outputs:
  for m inputs of a:
  \[ a.AF(\mu1, ..., \mu_m) = (\mu’1, ..., \mu’_m) \]
  where μk = μ(Pk) and μ’k = μ’(Pk) such that (Pk, k) ∈ IR, k = 1, ..., m and for n outputs of a:
  \[ a.AF(\mu1, ..., \mu_n) = (\mu’1, ..., \mu’_n) \]

A question may arise here about the selection of an activity among two or more enabled deterministic timed activities with equal completion time. The answer is to select one of them probabilistically. The selection probability for each activity is 1/n, where n is the total number of enabled deterministic timed activities with equal completion time.
where \( \mu_k = \mu(P_k) \) and \( \mu''_k = \mu'(P_k) \) such that \((a, k, P_k) \in OR, k = 1, ..., n \).

- **Step 2:** For any \( p \in SP \cup CP \):
  1. If \( p \in SP \),
     \( \mu(p) = \mu'(p) \).
  2. If \( p \in CP \) and \( p.SS = PRI \) and \( (p) \) is an input or output place of \( a \) or \( ((P_k, k, a) \in IR \) and \( P_k = p \) and \( k = 1, ..., m \) or \( ((a, k, P_k) \in OR \) and \( P_k = p \) and \( k = 1, ..., n) \) then
     \( \mu(p) = p.TL.\text{Sort}(p.\text{OF}(\mu')) \)
     Else
     \( \mu(p) = \mu''(p) \)

The final marking of the model after the completion of \( a \) is \( (\mu_1, ..., \mu_n) \), where \( \mu_k = \mu(P_k) \) and \( k = 1, ..., |SP \cup CP| \).

5. **Solution of OSAN Models**

In this section, we introduce methods for state space analysis and simulation of OSAN models.

5.1. **State Spaces of OSANs**

The standard method for the analysis of Petri nets is the analysis of state space or the reachability graph (RG) (also known as the occurrence graph [Jensen95]) of the model. The basic idea behind this method is to construct a directed graph, which has a node for each reachable system state and an arc for each possible state change.

**Definition 5.1**

State space (SS) of an OSAN model is defined as a 4-tuple \( SS = (M, E, A, M_0) \), where:

- \( M \) is the set of nodes (reachable markings of the graph) such that \( \forall \mu \in M: \mu = (\mu_1, ..., \mu_n) \), where \( \mu_i = \mu(P_i), i = 1, ..., n, n = |SP \cup CP| \) and \( P_i \in SP \cup CP \).
- \( E \) is set of arcs (edges) of the graph, and
- \( A \) is a node function such that:
  \( A: E \rightarrow M \times M \),
- \( M_0 \) is the initial marking of the OSAN model such that: \( M_0 = (\mu_1, ..., \mu_n) \), where \( \mu_i = \mu(P_i), i = 1, ..., n, n = |SP \cup CP| \) and \( P_i \in P \cup CP \).

As an example, we generate the state space of the OSAN model of Fig. 4. The initial marking of the model includes three coloured tokens \( A = \{a', 1, 1\} \), \( B = \{b', 1, 2\} \) and \( C = \{c', 2, 2\} \) in place SysBuf. Two other places (\( PBuf1 \) and \( PBuf2 \)) are empty. We represent the initial marking as \( ([CBA], [], []) \). In the initial marking, only the Route activity is enabled. After the completion of this activity, the marking of the net will change to \( ([CB], [A], []) \). In this marking, both Route and Proc1 activities are enabled. If the Route activity fire, the next marking will be \( ([C], [A], [B]) \). However, if the Proc1 activity fire, the next marking will be \( ([CA], [], [B]) \). The execution of the model continues and the set of all twelve reachable states that is generated by SANBuilder tool is shown in the reachability graph (RG) of Fig. 5.

Fig. 5. State space of the OSAN model of Fig. 4

5.2. **Reduced State Space Methods**

The main problem with the analysis of RGs of OSANs is its explosion. In general, the marking of coloured places of OSANs are represented by an ordered list. Obviously, RGs of OSANs may become very large, even for small models. Therefore, new methods are needed for the generation, storage and analysis of the state spaces of OSAN models. The following ideas can be useful for this purpose:

- **Reduced state space generation techniques:** Several techniques have been introduced for the reduction of state spaces. These state space reduction techniques are categorized in [Lewis02] as: removing information, compression, compositional techniques, preprocessing, and partial state space exploration. In each category, several methods are introduced in the literature. The Replicate/Join constructs [Sanders01], the graph composition formalism [Stillman99] have been used in UltraSAN and Möbius modeling tools for the generation of reduced state spaces of SAN models. These two methods take the advantages of state lumping of Markov chains.
- **Symbolic marking method:** This method is used in GreatSPN tool for the analysis of SWN models. Symbolic marking are obtained by disregarding the identities of objects within the places of the net and...
considering only their number. In this method, an aggregated state space called *symbolic reachability graph* (SRG) will be generated. The SRG corresponds to a lumped version of the complete RG and this aggregation is reflected also at the level of the underlying Markov process [GreatSPN01].

### 5.3. State Space Analysis of HOSAN Models

An HOSAN model contains one or more super activities. If an HOSAN model is well-defined, it has an equivalent flat OSAN model. For the analysis of an HOSAN model, it is possible to transform it into an equivalent flat OSAN model. A substitution algorithm can be employed to flatten the HOSAN model. In each step of the algorithm, all super activities on the leaves of the graph will be substitute by their definitions. This will be repeated until the only node on the graph is the root node. The resulting model is a flat OSAN model.

The main disadvantage of the above flattening method is the explosion of nodes in the resulting model that may lead to the explosion of the state space of the model. There are a few techniques for constructing SAN models in a way, which avoid state-space explosion problem. A key benefit of HOSAN models is the possibility of automatic employment of such techniques by their modeling tools. A modeling tool for HOSANs can transform an HOSAN model into the following composition formalisms:

1. **Replicate/Join construct**: A composition technique for SANs is the *Replicate/Join construct* [Sanders91]. A possible way of the analysis of HOSAN models is to transform them into this construct. A modeling tool for HOSANs can check for a tree-like structure in the dependency graph of an HOSAN model. Then, it can automatically find *shared states* based on the fusion places and organize the model using Replicate and Join operations. For the solution of the resulting model, the technique proposed in [Sanders91, Stillman99] can be employed.

2. **Graph Composition formalism**: Another composition formalism, which has been proposed in the Möbius modeling framework for SANs and other models, is the *Graph Composition formalism* [Stillman99]. This formalism does not limit the model hierarchy to a tree-like structure and any arbitrary structure (such as ring or mesh) is possible. A modeling tool for HOSANs can check for the possibility of the usage of this formalism. Then, it can transform the corresponding HOSAN model and uses the method proposed in [Stillman99] to solve the model.

### 5.4. Simulation of OSAN Models

Analytic solution of CTMCs could not be always used even for ordinary SAN models. Since, if a SAN model includes a non-exponential timed activity or its state-space is infinite, it may not be solved analytically by transformation to 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. In general, simulation can be used for the solution of OSAN models.

A discrete-event simulation algorithm for a flat OSAN model may have the following steps:

1. Determine the set of enabled activities in the current marking of the model,
2. Reactivate those disabled activities whose reactivation predicates are true in the current marking,
3. Generate the activity execution time for newly enabled or reactivated activities,
4. Apply the corresponding enabling rates on the remaining time of the enabled timed activities,
5. If more than one instantaneous activity is enabled in an unstable marking, select one of them probabilistically. The probability of selection of each enabled instantaneous activity \( a \) in marking \( \mu \) is computed by the formula (3.1).
6. In a stable marking (no instantaneous activity is enabled), considering the remaining time of all enabled timed activities and their respective enabling rates, select the next timed activity for completion. In the case of two equal completion times, select one of the corresponding timed activities, probabilistically,
7. Fire the selected instantaneous or timed activity in two steps:
   1. Execute the activity function, then
   2. Execute the reordering function of those coloured places, which has state-dependent order function.
8. Disable those enabled activities whose enabling predicate are not true in the current marking,
9. Set the simulation clock to the time of the most eminent event,
10. Collect the aggregated statistics and update the user-defined queries,
11. If the specified confidence level has not been achieved, go to step 1.

The main problem with simulation of large and complex models is the time needed to run the model. Several techniques, such as the work of [Mortensen01], [Sanders00] or [Gaeta96] have been introduced for efficient simulation of CPN models. In
techniques for fast simulation of hierarchical SAN models constructed with Replicate/Join formalism are introduced. These methods can be employed for OSAN models.

6. A Modeling Tool for OSANs

Modeling and analysis with Petri nets and their extensions need a software tool to help model construction and solution. We have developed a modeling tool for SAN-based models called SANBuilder [Abdollahi94a]. In addition to OSANs, this tool supports a new definition of SANs [Movaghar01], hierarchical stochastic activity networks (HSANs) [Abdollahi03a] and coloured stochastic activity networks (CSANs) [Abdollahi03b, Abdollahi04c]. A view of SANBuilder is shown in Fig. 6.

![Fig. 6. A view of the user interface of SANBuilder](image)

This tool provides an integrated development environment (IDE), which allows construction, compilation, animation, solution and simulation of SAN-based models in a single graphical user interface.

Main features of SANBuilder are as follows:

1. **The Graphical User Interface (GUI):** The GUI of SANBuilder is an IDE for editing SAN-based models, checking their syntax and semantics, compilation, animation and evaluation using state space analysis or simulation methods. This integration makes SANBuilder a user-friendly modeling tool.

2. **SAN Editor:** This graphical editor allows the construction of SAN-based models. An important feature of SAN Editor is its feature for saving and loading models and super activities in an interchange format, called SAN markup language (SANML) [Abdollahi04b]. SANML is based on the Petri nets markup language (PNML) [Billington03].

3. **Model Compiler:** The model compiler translates the model and then generates an executable model. It partially uses the compiler of C++ language to compile the model.

4. **Model Flattener:** A simple approach for the evaluation of hierarchical models (HSANs, HCSANs or HOSANs) is transforming them into an equivalent flat model and using the existing methods. Model Flattener does this transformation.

5. **Model Transformer:** An advanced approach for the evaluation of hierarchical models is using the existing techniques for the construction of composed models with reduced state spaces. Model Transformer converts a hierarchical model using one of these techniques (such as Replicate/Join [Sanders91] or Graph Composition Formalism [Stillman99]).

6. **Model Animator:** SANBuilder can animate models. A feature that is also known as token-game animation. For animating, the tool shows a step-by-step execution of the network by firing enabled activities and changing the marking. Animation is done in a stochastic manner (i.e. the result of discrete-event simulation of the model is animated on the graphical user interface). Debugging is a useful feature of this animator, which enables the modeler to define queries about the model behavior and trace their results while the animation is in progress.

7. **State Space Generator:** It executes a SAN model for generating its state space. The reachability graph (RG) of the model is built using the following State Space Handler module.

8. **State Space Handler:** This module stores and manages the RG in a hash table data structure.

9. **Analytic Solver:** If all timed activities of the model are exponentially distributed, this module can be used for model solution. If so, the corresponding continuous-time Markov chain (CTMC) will be generated and if it is finite, its ergodicity will also be checked. If it is ergodic, its global balance equations will be solved to compute its steady-state probabilities. Finally, the predefined results of the solution and user-defined queries will be computed from the steady-state probabilities and displayed on GUI.

10. **Discrete-Event Simulator:** If the steady-state solver cannot be used, simulation may be utilized for performance evaluation. This module simulates SANs using a discrete-event simulation algorithm. It also collects the predefined and user-defined statistics and displays them whenever the simulation is in progress or completed.
11. Automatic Code Generation: This feature allows the modeler to generate C++ source codes from the model of the system. We are working to add the capabilities for source code generation of more languages, including Java and Delphi.

7. Case Study: Transformation and Evaluation of UML Diagrams by OSANs

The unified modeling language (UML) is the de facto industrial standard of an object-oriented modeling language [Engels00]. The UML was developed as a general-purpose language, which consists of several sub-languages appropriate for modeling structural and behavioral aspects of a software system. The formal semantics for the notations included in the UML are not provided, which is the key requirement for the verification and evaluation purposes. To solve this problem, Petri nets formalism has been used as a complement to the UML in several research projects.

In this section, after a survey on related work, we will present two examples of the application of OSANs on modeling and evaluation of software systems. In this case study, we have transformed the UML use case and activity diagrams into OSAN models. Then, we have used SANBuilder tool to evaluate the resulting OSAN models.

7.1. Related Work

The following are some projects, which have used different extensions of Petri nets as a complement or alternative of UML for modeling and evaluation of software systems:

- In [Elkoutbi98], the use case and sequence diagrams are transformed to hierarchical coloured Petri nets (HCPNs).
- In [Saldhana00], UML diagrams are transformed to object Petri nets (OPNs).
- In [Pettit00], the dynamic behavior in UML is modeled and validated using coloured Petri nets (CPNs).
- In [Maier01], object coloured Petri nets (OCP-Nets) are used as a formal technique for object oriented modeling. In this work, use case diagram is transformed into OCP-Net models.
- In [Baresi02], UML models are formalized with OPNs.
- In [Eshuis02] the semantics and verification of UML activity diagrams for workflow modeling is done using CPNs.
- In [Jorgensen02], CPN models are used in an UML-based software development for designing a middleware for pervasive healthcare system.
- Derivation of an executable generalized stochastic Petri nets (GSPN) model from a description of a system, expressed as a set of UML state machines (SMs) is reported in [Merseguer02].
- The use of sequence diagram and statechart of the UML for the validation and the performance evaluation of systems is presented in [Bernardi02a]. For this purpose, the sequence diagrams are automatically transformed into GSPN models.
- [Bernardi02b] is focused on: (1) construction of stochastic Petri net models for dependable automation systems from a set of UML class diagrams and (2) translation and analysis of UML behavioral diagrams into analyzable GSPN models.
- Finally, in [Shin03], the UML-based system model is transformed to the CPN model using Design/CPN tool for validating the system behavior.

7.2. Transformation and Evaluation of Use Case Diagram by OSANs

One of the most important parts of the UML is its use case diagram. It is a useful tool for the requirement analysis of a system. Use case diagrams model the functionality of the system using actors and use cases. Use cases are services or functions provided by the system to its users. A use case is made up of three parts:

1. A natural description with possible alternative flows,
2. A diagram showing its objects, and optional
3. An interaction diagram to model its dynamic behavior.

As an example, the use case diagram of an automatic teller machine (ATM) system is shown in Fig. 7. In this example, the use case Identify is responsible for the identification of customers. A customer can check its balance by the use case Balance. He/she can use Withdraw or Deposit use cases to withdraw or deposit some money from/to his/her account. The relation between the use cases Balance and Identify is of type include. Therefore, the customer should be identified by the ATM system for each checking account. On the other hand, the relation between Print and Balance is of type extend. It means that the customer may want to print a report of his/her balance on checking account. A customer may want to print his/her balance or not. However, to check the
balance, the identification of customer is mandatory. This shows the difference between these two types of relations.

In [Elkoutbi98] coloured Petri nets (CPNs) is used for transformation of use case diagrams. Here, we use OSANs for such a transformation of the use case diagram of an ATM system. Our approach is similar to the work of [Elkoutbi98].

The OSAN model equivalent to this diagram is shown in Fig. 8, which is consisting of the following places and timed activities:

<table>
<thead>
<tr>
<th>Place</th>
<th>Timed Activity</th>
</tr>
</thead>
<tbody>
<tr>
<td>i_i: init include</td>
<td>s_i: start include</td>
</tr>
<tr>
<td>i_r: running include</td>
<td>f_i: finish include</td>
</tr>
<tr>
<td>i_f: include finished</td>
<td>b_r: balance running</td>
</tr>
<tr>
<td>b_r: balance running</td>
<td>d_r: deposit running</td>
</tr>
<tr>
<td>w_r: withdraw running</td>
<td>b_r: balance finished</td>
</tr>
<tr>
<td>h_r: deposit finished</td>
<td>w_f: withdraw finished</td>
</tr>
<tr>
<td>p_r: print running</td>
<td>f_b: finish balance</td>
</tr>
<tr>
<td>f_d: finish deposit</td>
<td>f_w: finish withdraw</td>
</tr>
<tr>
<td>f_p: finish print</td>
<td>b_f: balance finished</td>
</tr>
<tr>
<td>d_f: deposit finished</td>
<td>w_f: withdraw finished</td>
</tr>
</tbody>
</table>

There are some instantaneous activities in the model, which have been omitted in the above list. They model the probability of the execution of use cases.

To evaluate the model we consider the following parameters:

1. During the execution of deposit, withdraw or balance use cases, print use case will be executed with probability 0.5.
2. The time required for the execution of balance is half of the time of the other use cases (that are equal to each other),
3. The probability of the selection of withdraw is two times of deposit and balance use cases, and
4. All timed activities are exponentially distributed.

As expected, since the probability of the selection of withdraw was two times of deposit, the probability for its execution is two times of deposit.

Now, we evaluate the model with five simultaneous customers. The result of the solution is shown in Tab. 5. Based on the results of Tab. 5, since the Identify should be executed before any other use case, it is executing most of the time.

Fig. 7. Use case diagram of an ATM system

Tab. 4. The result of the evaluation of model for one customer

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Prob.</th>
</tr>
</thead>
<tbody>
<tr>
<td>No use case is executing</td>
<td>0.27</td>
</tr>
<tr>
<td>include is executing</td>
<td>0.27</td>
</tr>
<tr>
<td>Balance is executing</td>
<td>0.13</td>
</tr>
<tr>
<td>deposit is executing</td>
<td>0.07</td>
</tr>
<tr>
<td>withdraw is executing</td>
<td>0.13</td>
</tr>
<tr>
<td>print is executing</td>
<td>0.13</td>
</tr>
<tr>
<td>Simultaneous execution of deposit and withdraw use cases</td>
<td>0.00</td>
</tr>
<tr>
<td>One case executing check processing</td>
<td>0.15</td>
</tr>
</tbody>
</table>

Tab. 5. The result of the evaluation of model for five customers
### 7.3. Transformation and Evaluation of Activity Diagram by OSANs

Activity diagrams illustrate the dynamic nature of a system by modeling the flow of control from activity to activity. An activity represents an operation on some class in the system that results in a change in the state of the system. Typically, activity diagrams are used to model workflow or business processes and internal operation.

A method for the transformation of the UML activity diagram into CPN models is introduced in [Eshuis02]. This approach with some small changes can be used with OSAN models. Based on this method, we consider two places corresponding to the initial and final states and a timed activity for each state of the activity diagram in the equivalent OSAN model. A fork in the activity diagram is modeled by an instantaneous activity with two output places and a join is modeled by an instantaneous activity with two input places.

In Fig. 10, the OSAN model corresponding to the activity diagram of Fig. 9 is presented. In this model, which is constructed by SANBuilder tool, some simple places are used to model states. If we intend to model the flow of data in the activity diagram, we can alternatively use coloured places.

Now, we evaluate the OSAN model corresponding to the activity diagram of the process complaint workflow. The model is evaluated by SANBuilder tool. For simplicity, we have supposed that all instantaneous activities have the same case probabilities. We have also supposed that all timed activities have the same exponential distributions and the time required for timeout is 20 times greater than other activities. Now, we will run the model for one and five simultaneous users. In the first case, only one instance of the activity diagram is executing. However, in the second case, five instances of the activity diagram will be executed simultaneously.

In Tab. 6(a), the result of the model solution for one user is shown. In Tab. 6(b), the results for five users are displayed. Since, all timed activities are exponentially distributed, we have employed the steady-state solver of SANBuilder tool to evaluate the model.
8. Advantages of OSAN Models

Similar to ordinary SANs, OSANs can be used for modeling and analysis of various kinds and different aspects of computer and communication systems. These models can be used to build both Markovian and non-Markovian models.

The object-orientation of OSAN models helps them to be more useful for modeling and evaluation of software systems and business processes. The existence of features like those of queueing systems will make them useful for modeling and analysis of communication and manufacturing systems. Future case studies are expected to show the benefits of the application of these models in different areas.

OSANs addresses all five restrictions of SANs, which we have mentioned in section 2 of this paper:

1. **OSANs are hierarchical models.** OSANs provide hierarchies by introducing the notion of super activity. Super activities encapsulate hierarchies and provide compositionality for these models in a natural way and similar to object-oriented programming languages. OSAN models represent the hierarchy of hierarchical systems in a natural and understandable manner. Especially for software systems, OSANs are more appropriate than SANs plus composition techniques (such as Replicate/Join [Sanders91]).

2. **OSANs allow incremental modeling.** It is possible to incrementally construct large OSAN models for complex systems by starting with simple super activities and easily replacing them with enhanced ones.

3. **OSANs encourage reusability of submodels.** The existence of super activity with well-defined interface makes it possible to construct and share a repository of reusable OSAN submodels.

4. **OSANs provide facilities for data manipulation.** In addition to the simple (black) tokens of Petri nets, OSANs have coloured tokens that are objects of a user-defined colour class. Coloured tokens are added and removed from coloured places. These coloured tokens and its properties and methods can be accessed read-only in predicates of activities or be manipulated by input/output functions of activities.

5. **OSANs are object-oriented models.** OSANs support some most important characteristics of object-orientation, including encapsulation, polymorphism and inheritance for colour classes and super activities.
In addition, OSAN models have the following advantages over the other high-level extensions of Petri nets:

- Similar to a new definition of SANs [Movaghar01], OSANs have three different settings: non-deterministic, probabilistic and stochastic. This makes it possible to use them for verification (i.e. model checking over state spaces), probabilistic verification, or evaluation (i.e. steady-state or transient analysis or simulation) of distributed real-time systems.

- The flexibility of having functions for activities allows constructing models that are more compact than other high-level Petri nets.

- It is possible to use any general distribution function for timed activities.

- It is possible to specify a scheduling strategy for coloured places of OSANs.

- It is possible to automatically select and employ an appropriate composition technique (such as Replicate/Join or Graph Composition formalism) for the analysis of hierarchical OSAN models by modeling tools for OSANs.

- C++ or Java can be used as inscription language for the definition of token classes, predicates and functions of OSAN models. The selection of inscription language is a tool specific feature. It will allow to automatically generating C++ or Java source codes from OSAN models. This is an important need for modeling software systems.

9. Conclusions

In this paper, we have presented the informal and formal definitions, behavior, graphical notations, an example and solution techniques of object stochastic activity networks (OSANs). OSANs are an object-oriented extension for stochastic activity networks (SANs).

We have introduced some methods for the solution and simulation of OSAN models. These methods are based on the state-of-the-art techniques for the state space analysis and simulation of SANs, CPNs and OOPNs.

We have also briefly introduced SANBuilder modeling tool for SAN-based models. This tool supports a new definition of SANs, HSANs, CSANs and OSANs. This tool provides an integrated development environment for construction, animation, simulation and analytic solution of SAN-based models.

Finally, as a case study, we have transformed the UML use case and activity diagrams into OSAN models. The derived models have been evaluated using SANBuilder modeling tool. The results of this study proved that OSANs can be used as a complement or even an alternative of the UML on modeling and evaluation of software systems. The object-orientation of OSANs and the flexibility of having functions for activities, which is a key benefit of OSAN models over other extension of Petri nets, make them more appropriate for these applications.

To continue the current research, we are planning to work on the following topics in future:

- Introduce efficient techniques for the reachability analysis and numerical solution of OSAN models,
- Introduce fast methods for simulation of OSAN models,
- Do more case studies, which use OSANs for modeling and analysis of real systems, and

8. References


