Derivation and Evaluation of OSAN Models from UML Models of Business Processes

Ali Kamandi¹
Department of Computer Engineering
Sharif University of Technology
Tehran, Iran

Mohammad Abdollahi Azgomi²
Department of Computer Engineering
Sharif University of Technology
Tehran, Iran

Ali Movaghar³
Department of Computer Engineering
Sharif University of Technology
Tehran, Iran

Abstract
The unified modelling language (UML) is a de facto standard for object-oriented modelling. However, the formal semantics for the notations included in UML are not provided, which are a key requirement for the verification and evaluation purposes. To solve this problem, Petri nets formalism has been used as a complement to UML in several research projects. However, there is not a complete transformation technique for all concepts and diagrams of UML to an extension of Petri nets. We have recently introduced object stochastic activity networks (OSANs). OSANs are a high-level modelling formalism that integrates object-orientation into stochastic activity networks (SANs), which are a powerful and flexible stochastic extension of Petri nets. In this paper, we present some transformation techniques for the most important concepts and diagrams of UML into OSANs. The resulting OSAN models can be used for the evaluation purposes by analytic or simulation methods. As a case study, we have used the proposed technique for transformation and evaluation of a use case diagram.

Key words: UML diagrams, Evaluation, Petri nets, Object stochastic activity networks (OSANs).

¹ Email: kamandi@ce.sharif.edu
² Email: azgomi@gmail.com
³ Email: movaghar@sharif.edu
1 Introduction

The unified modelling language (UML) is the de facto industrial standard of an object-oriented modelling language [18]. UML was developed as a general-purpose language, which consists of several sub-languages appropriate for modelling structural and behavioural aspects of a software system. The formal semantics for the notations included in UML are not provided, which is the key requirement for the verification and evaluation purposes. To solve this problem, Petri nets (PNs) [21] formalism has been used as a complement to UML in several research projects.

Stochastic activity networks (SANs) [19, 20] are a stochastic generalization of Petri nets [21]. SANs inherits most features from queueing networks that make it appropriate for performance and dependability evaluation. SAN models are more powerful and flexible than most other extensions of Petri nets, including notable models such as stochastic Petri nets (SPNs) [18] and generalized SPNs (GSPNs) [16]. These models have been used in various applications, mainly for dependability and performability evaluations. SANs are supported with several powerful modelling tools, such as UltraSAN [24] and Möbius [9], developed by the PERFORM group [0] at UIUC.

We have recently introduced a new extension for SANs, called object stochastic activity networks (OSANs) [3, 1, 2]. OSANs are a high-level modelling formalism that integrates object-orientation into SAN models.

As a modelling tool for OSANs, we have developed a plug-in for Together® [14]. Borland® Together is a well-known commercial case tool for UML [26]. We have selected Together because of its powerful features for adding new models and good support of Java [13]. Having Together and OSAN plug-in, it is possible to have OSANs as a complement of UML to model business processes. Using this tool, one can model the whole business process of a system with UML and then model and analyze its behavioural aspects with OSANs.

In this paper, we will present the transformation of the most important diagrams of UML into the OSAN models. The behavioural parts of UML diagrams, including the use case, sequence, statechart and activity diagrams and active and passive objects are the subject of our study. Then we will present the result of a case study on application of this transformation technique for the evaluation of use case diagram of a business process.

The organization of this paper is as follows. In section 2, SANs and OSANs are briefly introduced. Section 3 gives a survey on related works. In section 4, some transformation techniques of UML models into OSANs are presented. In section 5, as a case study, an example of the illustrated transformation techniques and the results of the model evaluation are presented. Finally, some concluding remarks are mentioned in section 6.

2 Object Stochastic Activity Networks

In this section we briefly introduce SANs, OSANs, their elements, graphical notations and methods and tools for their evaluation.
2.1 Stochastic Activity Networks

Stochastic activity networks (SANs) [20]: place, timed activity, instantaneous activity, input gate and output gate. Graphically, these primitives are represented as follows: a place is depicted as $\text{\ding{092}}$, an instantaneous activity is represented as $\text{\ding{094}}$, a timed activity as $\text{\ding{095}}$, an input gate with $n$ inputs is shown as $\text{\ding{098}}$ and an output gate with $m$ outputs as $\text{\ding{099}}$.

Places of SANs are equivalent to the places of Petri nets. Timed and instantaneous activities correspond to the timed and immediate transitions of GSPNs. Timed activities represent activities of the modelled system whose durations impact the system’s ability to perform. Timed activities represent parallelism in the modelled system. A timed activity has a number of inputs and outputs. To each timed activity is associated an activity time distribution function, an enabling rate function (which is the activity execution speed) and a reactivation predicate. On the other hand, 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 modelling nondeterminacy.

Gates model complex interactions among activities and, thus, increase the modelling flexibility. There are two kinds of gates: input gate and output gate. Input gate links a finite number of input places to an activity. To each such input gate is associated an enabling predicate and an input function. The enabling predicate can be used to specify complex enabling function, while input functions are used to change the state of the model. To be able to fire in a marking, the enabling predicate of all input gates of an activity must be true. On the other hand, output gate links an activity to a finite number of output places. To each such output gate is associated an output function, that is used to change the state of the model.

SANs are based on a unified view of the system in three settings: nondeterministic, probabilistic, and stochastic. The nondeterministic setting of SANs is referred to as activity networks, which are nondeterministic models for representing concurrent and reactive systems. Application of this setting is on the analysis of logical aspects or verification of concurrent and reactive systems. Disregarding the timing related information of the model and viewing it in a nondeterministic setting accomplish this. The activity network model is then translated into a transition system and verification is done.

For evaluating the operational aspects of systems such as performance, dependability and performability, the stochastic setting of SANs is used. In this setting, both nondeterminacy and parallelism are represented probabilistically [20]. Based on the probability distribution of timed activities, Markovian or non-Markovian SAN models will be resulted.

Application of the another setting of SANs, namely, probabilistic setting, is on probabilistic verification.

2.2 Elements of OSAN Models

The elements of OSANs are simple place, coloured place, instantaneous activity, timed activity, and super activity [1, 2, 3]. In OSANs, input and output gates of
SANs are encapsulated in instantaneous and timed activities and no longer exist. Gates in SANs defines the behaviour of the corresponding activities. Therefore, their encapsulation into activities is more meaningful. By this encapsulation, all elements of OSANs are defined as classes. The activities have some methods where one of these methods is its activity function that acts as input/output gate functions and changes the state of OSAN after completion of the activity. Another reason for this integration is to eliminate a non-standard element that does not have any equivalence in other standard extensions of Petri nets.

In addition to the simple place, which is equivalent to the places of SANs and Petri nets and holds tokens of integer type, OSANs have another kind of places called coloured place. A coloured place holds a list of tokens (objects) of a token class. Tokens are removed from a coloured place by activities based on the specified selection policy. The selection policy can be selected from the set \{FCFS, LCFS, PRI, NSP\}. If the NSP selection policy is selected, then the coloured place will hold an unordered list or a multiset of tokens. Tokens of OSANs are active objects, which have some fields (which represent state) and methods (which model behaviour). Methods of tokens are general functions (which may also be state-dependent).

Now we briefly introduce the elements of OSANs:

1. **Simple place**: A simple place in OSANs is same as SANs or Petri nets. Graphically, a place is drawn as a circle (\(\bigcirc\)). A simple place may contain zero or more tokens, which describe the state of a place. A token is drawn by a black dot inside the circle of a place, (ex. \(\bigcirc\)). Only the number of tokens in a simple place is used in the predicates and functions of timed or instantaneous activities.

2. **Token class**: Tokens of OSANs are objects of a previously defined token class. Each token class has one or more fields and zero or more methods. Token classes can be defined as a Java class. As we mentioned before, methods of tokens are general functions. These methods can be called by functions of activities. It means that the treatment of an activity can vary from one token to another, depending to their methods.

3. **Coloured place**: Graphically, a coloured place is drawn as an oval (\(\vadjust{\small{(TC,SP)}}\)), where \(TC\) is the name of a previously defined token class and \(SP\) is a selection policy, which can be selected from the set \{FIFO, LIFO, PRI, NSP\}. A coloured place holds a list of tokens of a specified token class. Both the number of tokens in a coloured place (the size of the token list) and the value of the fields of tokens (token values) can be used in predicates and functions of activities. To be enabled, an instantaneous or timed activity must have sufficient tokens on its input places and the colour of tokens or token values should match its enabling predicate. Tokens are removed from the token list based on the specified selection policy of the coloured place. This will limit the simultaneous remove of tokens from a coloured place to one at a time. If an activity needs to remove more than one token from a coloured place, it can call the remove function several times.

\(^4\)NSP stands for No-Selection-Policy
4. **Timed activity:** Graphically, a timed activity of OSANs with \( n \) inputs and \( m \) outputs, where \( m \geq 0 \), \( n \geq 0 \) and \( m+n > 0 \), is shown as \( \text{\textcircled{\textbullet}} \). A timed activity is linked to zero or more \((m)\) input places and zero or more \((n)\) output places. A timed activity of OSANs is an encapsulation of the timed activity, input gate and output gates of SANs. To each timed activity is associated an enabling predicate, an activity time distribution, an enabling rate function and an activity function (which defines the possible changes of the state of OSAN after completion of the activity).

5. **Instantaneous activity:** Graphically, an instantaneous activity of OSANs with \( n \) inputs and \( m \) outputs, where \( m \geq 0 \), \( n \geq 0 \) and \( m+n > 0 \), is shown as \( \text{\textcircled{\textbullet}} \). An instantaneous activity is linked to zero or more \((m)\) input places and zero or more \((n)\) output places. An instantaneous activity of OSANs is an encapsulation of the instantaneous activity, input gate and output gates of SANs. To each instantaneous activity is associated a case probability function and an activity function (which defines the possible changes of the state of OSAN after completion of the activity).

6. **Super activity:** A super activity is a submodel of OSANs, which has zero or more input/output fusion places. A fusion place can be a simple or coloured place. A super activity is composed as a hierarchy of some places, activities, and lower-level super activities, input and output fusion places. Super activities can be defined and stored separately. Previously defined and stored super activities may be used to compose a new OSAN model. In a graphical representation, fusion places are depicted as \( \text{\textcircled{\textbullet}} \) or \( \text{\textcircled{\textbullet}} \), corresponding to a simple fusion place or a coloured fusion place, respectively. A super activity with \( n \) inputs and \( m \) outputs, where \( m \geq 0 \), \( n \geq 0 \) and \( m+n > 0 \), is shown as \( \text{\textcircled{\textbullet}} \).

### 2.3 Evaluation of OSAN Models

Generally, methods for the solution of OSAN models is based on the existing methods for the solution of SANs and other high-level extensions of Petri nets. We have introduced methods and techniques for the solution and simulation of flat and hierarchical OSAN models \([3]\).

We have also developed a tool for OSANs as a plug-in of Together, which is a well-known tool for UML. A useful feature of Together is to allow the user to implement his/her own model in its environment. Using this feature, we have developed a plug-in for OSANs. After the installation of this plug-in, a user can make an OSAN model in Together's environment similar to diagrams of UML. After the construction of an OSAN model, it is possible to call an appropriate solution technique to simulate, animate or solve it through Together's pop-up menu. Having both UML and OSANs in a same tool will facilitate constructing models for business processes with UML and then model and evaluate the behavioural aspects with OSANs.

A view of the user interface of Together with OSAN plug-in is displayed in Fig. 1.
The most useful features of this plug-in are as follows:

- Editing OSAN models in Together's graphical user interface,
- Load and save OSAN models and super activities as separate files,
- Having both UML and OSAN models in separate tab pages,
- Flattening hierarchical OSAN models,
- Animation (state-by-step execution) of OSAN models,
- State space generation for OSAN models,
- Analytical solution of OSAN models,
- Discrete-event simulation of OSAN models,
- Allowing user to define queries about the behaviour of the model,
- Evaluating the user-defined queries, and
- Displaying the result of the model solution and simulation.

### 2.4 Some Advantages of OSAN Models

OSAN models have the following advantages over the other high-level extensions of Petri nets:

1. **OSANs** are object-oriented and stochastic models. Both model structure and token definitions of OSANs are object-oriented.

2. The existence of non-deterministic, probabilistic and stochastic settings for these models makes them useful for functional analysis (i.e. verification over state spaces) as well as operational analysis (i.e. evaluation by simulation or steady-state or transient solution techniques).

---

Some of the mentioned advantages are inherited from SANs.
3. The flexibility of having functions for activities, which can generally change the states of an OSAN will facilitate constructing compact models.
4. It is possible to use any general distribution function for timed activities.
5. Super activity of OSANs is a key technique for achieving to compositionality and reusability of these models.
6. Place fusion is a natural way for composing hierarchical models in OSANs.

3 Related Works

Petri nets formalism has been used as a complement to UML for validating the behaviour of systems in several research projects. In [8], the use case and sequence diagrams are transformed to hierarchical coloured Petri nets (HCPNs). In [23], UML diagrams are transformed to object Petri nets (OPNs). In [22], the dynamic behaviour in UML is modelled and validated using coloured Petri nets (CPNs). In [15], object coloured Petri nets (OCP-Nets) are used as a formal technique for object oriented modelling. In this work, use case diagram and diagrams is transformed into OCP-Net models. In [3], UML models are formalized with OPNs. In [11] the semantics and verification of UML activity diagrams for workflow modelling is done using CPNs. In [12], 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 [17]. The use of sequence diagram and statechart of UML for the validation and the performance evaluation of systems in presented in [5]. For this purpose, the sequence diagrams are automatically transformed into GSPN models. [6] 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 behavioural diagrams into analyzable GSPN models. Finally, in [25], UML-based system model is transformed to the CPN model using Design/CPN tool for validating the system behaviour.

4 Transformation of UML Diagrams into OSANs

In this section, we present the transformation of the sequence, use case, statechart and activity diagrams and active and passive objects of UML to OSAN models. The purpose of this study is to investigate the possibility of using OSANs as a complement to UML for formal modelling the behaviour of systems. The derived models can be used for the analysis of the logical or operational aspects, using the verification or evaluation techniques. The flexibility and object-orientation of OSANs make these models more useful for such applications.
4.1 Sequence Diagram

A sequence diagram [7] or interaction diagram of UML is a semi-formal technique and lacks adequate means to model concurrency [15]. In [8] CPNs and in [15] OCP-Nets are used to replace this diagram. In [6], sequence diagrams are also automatically transformed into GSPN models.

In this section, we will model the sequence diagram of an automatic teller machine (ATM) system by OSAN models. As an example, we use the sequence diagrams of the ATM system in [8], corresponding to the use case Identify. These sequence diagrams are shown in Fig. 2. One of these diagrams is used when the PIN entered by the customer is valid and the identification will be successfully done. The second sequence diagram is used when the PIN is invalid. In [8], a table of object states is derived from the states of the system after the execution of each step of the sequence diagram. This table of object states is used to derive a CPN model. We have used a same method to derive two OSAN models for the sequence diagrams corresponding to the use case Identify, which is shown in Fig. 3.

Based on the method of [8], we should integrate two models to derive an OSAN model for the execution of the use case Identify. The resulting model is shown in Fig. 4.

Fig. 2. Two sequence diagrams corresponding to the use case Identify [8], (a) user enters valid pin and (b) user enters invalid pin.

Fig. 3. OSAN models for the sequence diagrams corresponding to the use case Identify, (a) user enteres valid pin and (b) user enters invalid pin

Fig. 4. OSAN model for the integrated sequence diagrams for the use case Identify
4.2 Use Case Diagram

One of the most important parts of 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 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 behaviour.

As an example, the use case diagram of an ATM system is shown in Fig. 5. 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 [8] coloured Petri nets (CPNs) and in [15] the object coloured Petri nets (OCP-nets) are used for the use case modelling. In this section, we use OSANs for such a transformation of the use case diagrams.

Our approach is different from the above mentioned works. We will transform the use case diagram in several steps. Each use case is realized with a sequence or collaboration diagram. Therefore, the transformation of relations of type generalization is postponed until the transformation is done for one of these diagrams.
Step 1. Transformation of separate use cases. We consider each use case separately and transform it into an OSAN model. In this transformation, we use two coloured places. Coloured place initial as the starting point and coloured place running representing the execution state of the use case. The number of tokens in the initial place represents the number of simultaneous execution of instances of the use case. A timed activity, start, is used to model the rate and time of the execution of use cases. The timed activity finish models the time required for the execution of the use case. The OSAN model for this step is displayed in Fig. 6.

Step 2. Transformation of the extend relation. Now we transform the extend relation to an OSAN model. The resulting model is displayed in Fig. 7. In this figure, when the use case uc1 is executing, the execution of the use case uc2 is optional. The timed activity finish_uc1_start_uc2 decide based on the conditions indicated in its activity function.

Step 3. Transformation of the include relation. Based on the semantics of the include relation, the execution of uc2 is mandatory. Several scenarios are assumed for the execution of the use cases uc1 and uc2:
1. First uc1, then uc2,
2. First uc2, then uc1,
3. uc1 partially, then uc2, finally the rest of uc1.

For the first and second scenarios, the corresponding OSAN model can be derived in this step. However, for the third scenario, the realization is needed. The resulting OSAN model for the first scenario is displayed in Fig. 8.

Step 4. Realization of use cases. For the realization of use cases, the sequence diagram is used. We presented the transformation of sequence diagram in the previous section. Here, we use this transformation technique with the following considerations:
• All places are coloured,
• There is an OSAN model for each scenario,
• In each model, names of the token classes of coloured places are corresponding to their scenarios (ex. uc1_scenario1 and uc1_scenario2),
• uc1_scenario1 and uc1_scenario2 classes are inherited from uc1_scenario,
• The two scenarios are integrated using the previous method.

Step 5. Refinement of the model. After we modelled the use cases and their scenarios, we should refine the resulting model based on include and generalization relations. The concept of generalization is presented in Fig. 9. In this figure, the scenario of the child use case is as follows. Some parts of its scenario may be executed, then some parts of the scenario of its parent will be executed, then some parts of its scenario, and so on.

In the OSAN model of the scenario for a use case, whenever the execution is transferred to another use case there will be a timed activity in the model (a parent use case in the generalization or an included use case for the include relation). The OSAN models corresponding to the execution of a parent or child use case is shown in Fig. 10.

Now, we integrate two OSAN models of the parent and child use cases into the OSAN model of Fig. 11. This is done based on the method described in the
previous section. For the *include* relation, the execution will transfer one time to the included use case. However, for the generalization, it may be repeated several times.

![Fig. 6. OSAN model corresponding to the first step of the transformation of use case diagram](image)

![Fig. 7. OSAN model for the extend relation](image)

![Fig. 8. OSAN model for the *include* relation (the first scenario)](image)

![Fig. 9. Execution of a scenario with generalization relation](image)
4.3. Statechart Diagram

The behavioural specifications in UML are based on statechart. Statechart describes all of the possible states that a particular object can get into and how its state changes as a result of events that reach the object. Statechart diagrams specify the sequences of states. These diagrams are especially useful in modelling reactive objects whose states are triggered by some specific events.

In [23], a method for deriving an object Petri net model (OPM) for statechart diagram is introduced. In this method, the statechart diagrams are first transformed into OPM. Then, using the interaction diagram, the derived models are combined and the integrated model is derived.

We use a similar method to transform statechart diagram into OSAN models. The transformation of simple statechart diagrams with no composite state is simple. For this purpose, each state will be mapped into a simple or coloured place and each transition to a timed activity in the corresponding OSAN model. If there are composite states in the statechart diagram (which means that the...
diagram is nested or hierarchical), it is required to flatten the diagram before its transformation. A composite state includes some sequential or parallel substates.

In Fig. 12, the OSAN model of a statechart diagram with concurrent composite states, which is explained in [23], is shown. Fig. 12(a), with concurrent composite states is flattened into the diagram of Fig. 12(b), which fork and join states are added to model the concurrency. "A fork state splits an incoming transition into two or more transitions. A join state merges several transitions" [23]. Fig. 12(c) is the equivalent OSAN model. In this model, fork and join states are mapped into activities. Since, Petri nets (and therefore, OSANs) are inherently models for concurrent systems, fork and join states are not required in the converted OSAN models.

4.4. Activity Diagram

Activity diagrams illustrate the dynamic nature of a system by modelling 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 UML activity diagram into CPN models is introduced in [11]. This approach with 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 modelled with an instantaneous activity with two output places and a join with an instantaneous activity with two input places.

In Fig. 14, the OSAN model corresponding to the activity diagram of Fig. 13 is presented. In this model, simple places are used. If we want to model the flow of data in activity diagram, we can alternatively use coloured places.

4.5. Active and Passive Objects

Active objects are the basis of concurrency in UML models. Active objects may operate asynchronous or periodic (synchronous). The activation of an asynchronous active object may be done by interrupt or event. However, a periodic active object may be activated with a timer event. In [22] a method to derive CPN models for active and passive objects is presented. We will adapt this method with OSANs.
Fig. 12. Conversion of concurrent composite states X to a flat OSAN model

Fig. 13. Activity diagram for "Processing complaints" workflow [11]

Fig. 14. An OSAN model equivalent to the activity diagram of Fig. 13
4.5.1. **OSAN model of asynchronous active objects.** In the OSAN model of an asynchronous active object, there are some places and timed activities, which a control token is responsible for the presentation of the control flow of the object. In Fig. 15, an OSAN model corresponding to an asynchronous active object is shown. In this model, a token in AsyncControl place represents a situation that an object is ready to receive an interrupt. In this situation, if the input event occur, ProcessEvent timed activity will be fired. After this occurrence, the object will not be able to reply to any other event. After the processing of input event, SendMsg activity will be fired and the control token will be added to AsyncControl place. Then the object will be enabled to receive and process another event. It can also generate and send some events to other parts of the system.

An active object is a part of a system. Therefore, we have modelled it as a super activity. As we mentioned in section 2, a super activity is a submodel of OSANs. There are two input and output fusion places named InputEvent and OutputEvent in the super activity. These fusion places will be bound to two actual places in the OSAN model of the whole system.

4.5.2. **OSAN model of periodic active objects.** A periodic active object is enabled periodically by a timer. The OSAN model for a periodic active object is similar to that of an asynchronous active object. In the OSAN model of Fig. 16, a place (Sleep) and an activity (Wakeup) are added. Sleep and Wakeup will cause some periodic delay. This will make the object to be ready for receiving events. The period of delay will be determined by the time distribution of the Wakeup timed activity.

4.5.3. **OSAN model of passive objects.** A data object, which maintains and provides access to the data, is an example of a passive object. If we suppose that access to the data is exclusive, the method proposed in [22] can be employed. For each read and write operation on the object, an activity and two simple and coloured places are required. The simple place represents the request for the operation and the coloured place will contain the result of the read operation, or the returned value of the write operation.

Fig. 15. An OSAN super activity corresponding to an asynchronous active object

Fig. 16. An OSAN super activity corresponding to a periodic active object
5 Case Study

In this section, we will present an example of the application of our transformation technique for use case diagram of a business process. We have used the Together plug-in for OSANs to construct and evaluate the models.

5.1 OSAN Model for the Use Case Diagram of an ATM System

The first example is the OSAN model equivalent to the use case diagram of an ATM system. The OSAN model is shown in Fig. 17. The places and timed activities used in the model are listed in Table 1. There are also some instantaneous activities in the model, which model the probability of the execution of use cases. These instantaneous activities are not listed the table.

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.

Fig. 16. OSAN model corresponding to the use case diagram of an ATM system

<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: finished</td>
<td>f_b: finished balance</td>
</tr>
<tr>
<td>b_r: balance running</td>
<td>f_d: finish deposit</td>
</tr>
<tr>
<td>d_r: deposit running</td>
<td>f_w: finish withdraw</td>
</tr>
<tr>
<td>w_r: withdraw running</td>
<td>p_r: print running</td>
</tr>
<tr>
<td>b_f: balance finished</td>
<td></td>
</tr>
<tr>
<td>d_f: deposit finished</td>
<td></td>
</tr>
<tr>
<td>w_f: withdraw finished</td>
<td></td>
</tr>
</tbody>
</table>

Table 1
Places and timed activities used in the model
5.2 The Results of the Model Evaluation

The result of the solution of the model, if there is only one customer in the system, is shown in Table 2 (a). Since, all timed activities have exponential activity time distributions we have employed the steady-state solver of the tool to evaluate the model. We can alternatively simulate the model.

As it was 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. 2 (b). As it is appeared from the results, because the Identify should be executed before any other use case, it is executing most of the time.

The model displayed in Fig. 16 is quite flexible. For example, changing the model for 100 customers is simply possible.

<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>0.065</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>0.0</td>
</tr>
<tr>
<td>One case executing check processing</td>
<td>0.15</td>
</tr>
</tbody>
</table>

Table 2

The result of the solution of the OSAN model of the use case diagram of an ATM system: (a) One customer in the system and (b) Five customers in the system

6 Conclusions

In this paper, we have presented the transformation of the most important diagrams and concepts of UML into object stochastic activity network (OSAN) models. The purpose of this transformation is to use the resulting OSAN models in a modelling tool, like our plug-in for Together, for the evaluation and verification of the behaviour of business process models.

The object orientation of OSANs and the flexibility of having input/output methods for activities, which is a key benefit of OSAN models over other extension of Petri nets, make them more appropriate for these applications. The existence of a tool, which is integrated into a well-known tool for UML, is another benefit of using OSANs instead of the other extensions of Petri nets for this purpose.

We have also presented some examples of the application of our transformation techniques. We have evaluated the resulting OSAN models by our plug-in for Together.

We intend to work on the following related topics in future:
- Consider all concepts of UML for transformation,
- Formally define the transformation from UML to OSANs,
• Develop a tool for automatic transformation from UML to OSANs, and
• Do more case studies.

References


Online References:

26. URL of PERFORM group of UIUC: www.crhc.uiuc.edu/PERFORM

27. URL of Borland Together: www.borland.com/together