Chapter 5
Model Checking State Machines Using Object Diagrams

Thouraya Bouabana-Tebibel

Abstract UML behavioral diagrams are often formalized by transformation into a state-transition language that sets on a rigorously defined semantics. The state-transition models are afterwards model-checked to prove the correctness of the models construction as well as their faithfulness with the user requirements. The model-checking is performed on a reachability graph, generated from the behavioral models, whose size depends on the models structure and their initial marking. The purpose of this paper is twofold. We first propose an approach to initialize formal models at any time of the system life cycle using UML diagrams. The formal models are Object Petri nets, OPNs for short, derived from UML state machines. The OPNs marking is mainly deduced from the sequence diagrams. Secondly, we propose an approach to specify the association ends on the OPNs in order to allow their validation by means of OCL invariants. A case study is given to illustrate the approach throughout the paper.

Introduction

Formalisms integration is a key concept in software engineering. It enhances the development process quality and ensures its reliability. Often, when complex systems need to be studied with regard to various aspects, it doesn't make it easy to find a unique formalism supporting all the aspect constructs and their related semantics. One used technique is to integrate two or more formalisms to accurately specify each aspect. The provided specification is thus constructed by integration, in a complementary way. It must satisfy completeness and consistency properties. It also must rely on a well defined semantics allowing a formal verification. On

T. Bouabana-Tebibel (✉)
Laboratoire de Communication dans les Systèmes Informatiques - LCSI, Ecole nationale Supérieure d'Informatique - ESI, Algiers, Algeria
e-mail: ttebibel@ini.dz

the other hand, for formalization and verification purposes, informal specifications are often transformed into specifications whose formalism is chosen according to the numerous verification mechanisms it supports. We talk here about formalisms integration per derivation.

The purpose of this paper is to verify UML [27] modeling by deriving the constructed diagrams into OPNs [21]. In UML, data initialization is provided by means of object diagrams specifying the object identity, its attribute values as well as its state at the time of initialization. Objects state can be omitted when, by default, all data are given for the initial state of the system life cycle. However, in software engineering, some systems are sometimes studied beginning from a state that is different from their initial state. Indeed, when a system already exits, and designers only project to update, restructure or extend some of its functionalities, just a part of its life cycle needs to be revised or added. In these cases, some of the system objects will move from their initial state whereas others have already moved through their life cycle and so are located on states that are different from the initial one at the moment of the analysis. The object life cycle may be described by means of a state machine, in case of UML modeling, or by an object Petri net (OPN) if a formal specification is provided. In the latter case, allowing an OPN marking at the places translating the appropriate time, not necessarily the initial one, will better describe the real-world system without need to rework the unchanged object models. Just the initial marking of those OPNs has to be reset when new and revised object models are constructed and then connected to the existing OPNs. The OPNs modular architecture appears to be especially convenient for this kind of deployment. It makes it easy to execute new systems considering changes only on the OPNs marking. Another advantage behind starting a system behavior at a time different from the initial one is to reduce the accessibility graph size which will be truncated of all the states space preceding the new system starting. Reduction of the accessibility graph size prevents a combinatory explosion of its states.

The key idea of the present contribution focuses on the relevance of the association ends, specified on the class and object diagrams, regarding the information they provide to deal with a model analysis. We will show how this information can be used in the validation process to check the models correctness. So, we firstly propose to mark the OPNs, derived from state machines, at a specific point in time, with objects extracted from the object diagram. The association ends specified on the object diagram will also provide the OPNs with marks representing objects with specific roles at a given time of the system lifecycle. The marks are composed of object identities and attribute values. The initial marking approach proposed in this work provides the possibility of lunching the model checking at different states of the system life cycle without need to revise the OPNs.

The other results we propose regarding the association ends concern the way they will be specified on the OPNs in order to allow the checking of OCL invariants transformed into temporal logics. This specification is derived from link actions described on the state machine. In fact, as long as OCL navigation expressions are not used, the association end specification onto the object life cycle is not required.

Otherwise, this specification provides after transformation into OPNs a formal basis for the validation of the OCL invariants.

The remainder of the paper begins with a brief presentation of the state machines formalization work we published in [8]. We show in section “Background” the novelty and relevance of this work by comparison with related works. In sections “Association End Specification” and “Initialization Approach” the proposed approach is presented and the techniques on which it rests are developed. This approach is validated in section “Validation of the Approach”. We conclude with some observations on the obtained results and recommendations for future research directions.

Related Works

Many works [10, 17, 19, 22, 23, 29, 30] proposed a denotational semantics to the notation by projecting it in a rigorously defined semantic domain. Some studies have already addressed the formalization of UML behavioral diagrams by translation into OPNs semantics domain. The most known are those of Baresi. He proposed in [4] a textual and graphical formalization of some UML behavioral specifications using OPNs. He afterwards reinforced his proposal in [3] by defining translation recommendations. He only achieved the formulation of formal conversion rules for syntactic models in [5]. The drawback of this work is the constraint of writing the UML models in a canonical language called LEMMA. More recently, he formalized in [6] some constructs of the interaction overview diagram using a temporal logic called TRIO. The proposed semantics was implemented in the Zot tool to prove some user-defined properties. Contrary to our approach, the properties are written in a generic manner abstracting the object values.

Bokhari and Poehlman offer in [7] to transform UML state machines in OPNs in order to analyze them. The model validation resulting from the derivation is performed on the model checker DesignCPN. No details are however given about the initialization of the model that deals with identified objects. Similarly, Hsiung et al. presented in [20] an approach for the formalization of statecharts with colored Petri nets. For this purpose, they use sequence diagrams to initialize their models and OCL constraints transformed into temporal logic to validate them. But the model initialization starts from time zero. Other authors, as Harel, establish a strong relationship between state machines and sequence diagrams. In [18] Harel et al. describe a methodology for synthesizing statechart models from scenario-based requirements. The requirements are given in the language of live sequence charts (LSCs), and may be played in directly from the GUI. The resulting statecharts are of the object-oriented variant, as adopted in the UML. Besides its theoretical interest, this work also has practical implications, since finding good synthesis algorithms could bring about a major improvement in the reliable development of complex systems.

In [16] the basic structure of UML sequence diagrams is first analyzed and then their formal description using OPNs is given. For reuse, the formal description of reusable interactions is studied. Next, the authors put forward the mapping algorithm of UML sequence diagrams into OPNs, which ensures the accuracy, integrity and simplicity of the results by four steps, including abstraction, merging, synchronization and reduction. This approach provides a good foundation for automatic verification except that the only considered starting time is the beginning of the system life cycle.

Fish and Störrle offer in [13] a number of principles applicable to visual languages characterized by imprecise semantics in order to analyze and discuss their quality. Based on this approach, they identify many sources of potential errors in UML diagrams and propose solutions to these deficiencies.

New approaches of the UML formalization techniques are graph transformation [19] and more recently, grammar graphs [24]. These techniques give more precision to the UML diagrams semantics without the use of formal languages. Holscher et al. propose in [19] to integrate UML diagrams, namely use case, class, object and statechart diagrams, into a graph transformation system. They afterwards provide rules to change the system states. To construct the first state, they were faced with the issue of retrieving the appropriate state of each modeled object. To achieve this purpose, they constrain the modeler to specify for each object on the object diagram its current state. The approach we propose removes this constraint by only using the association ends specification. In [24], a graph grammar is automatically derived from a state machine to summarize the hierarchy of the states. Based on the graph grammar, the execution of a set of non-conflict state transitions is interpreted by a sequence of graph transformations.

More recent works tackle the formalization of the interaction overview diagram which integrates sequence diagrams, thus providing some data to the behavioral models. But none focuses on the use of valuated objects to mark the targeted formalisms. Andrade et al. formalize in [1,2] the interaction overview and sequence diagrams by means of Time Petri Nets to analyze and verify embedded real-time systems with energy constraints. The approach resorts to the use of annotations provided by the MARTE UML profile for verifying qualitative properties as time and energy savings. To improve the formalization, the transformation needs to be automated and the models hierarchically structured. In [22] the interaction overview diagram semantics is formalized by the stochastic process algebra PEPA where the sequence diagrams are abstracted by colored tokens. This work is completed in [11] to analyze the modeling. But contrary to ours, the analysis is restricted to generic models.

Regarding the association ends, no works tackle their integration within state machines. We can explain this arguing that the UML/OCL association is rarely used to formally validate the UML models. When done, it is limited to OCL invariants handling only attribute expressions [32] or OCL pre and postconditions [14, 15]. Generally, the formalized UML models are rather coupled with formalisms for the expression of system properties.

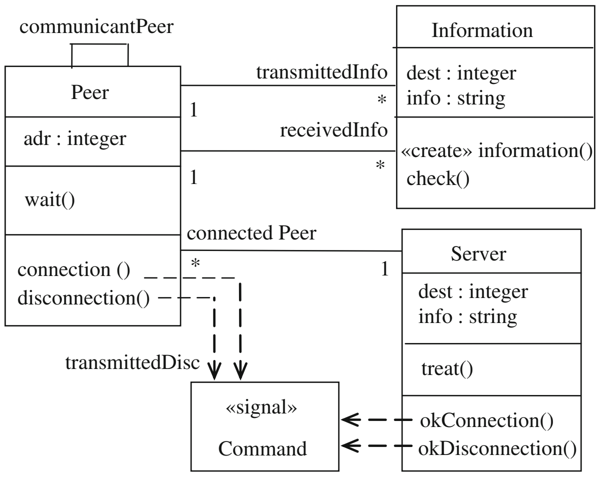


Fig. 5.1 Peer to peer class diagram

The work we are presenting in this paper brings a new contribution in the field of UML formalization. It extends the approach that we proposed in [9] by focusing on the relationship between sequence diagrams and state machines. It also proposes an approach to specify the association ends on the OPNs in order to deal later with the models validation.

Background

We present in this section the main results obtained after transformation of state machines into OPNs. This approach was developed in [8].

Case Study

To illustrate the transformation mechanisms and those proposed in this paper, we take a case study on a brokered peer to peer system. The main activity of this system is the information exchange between the peers after they have been identified by the server. Identification is established after a connection request confirmed by the server. Once connected, peers interact by exchanging information. Figure 5.1 shows the class diagram of the application, illustrating the various system objects and their actions.

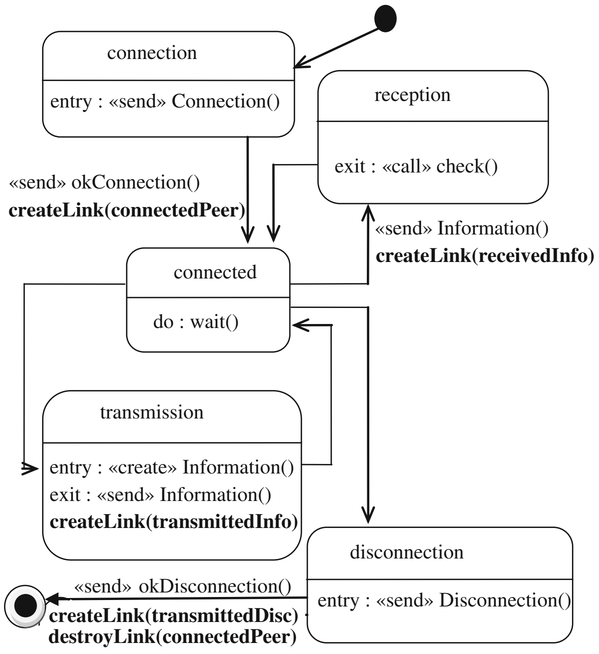

connection
entry : «send» Connection()
reception
exit : «call» check()
«send» okConnection()
createLink(connectedPeer)
«send» Information()
createLink(receivedInfo)
connected
do : wait()
transmission
entry : «create» Information()
exit : «send» Information()
createLink(transmittedInfo)
disconnection
entry : «send» Disconnection()
«send» okDisconnection()
createLink(transmittedDisc)
destroyLink(connectedPeer)

Fig. 5.2 State machine of a peer

Transforming State Machines to OPNs

A state machine [27], noted SM in the following, formally describes the behavior of objects of a given class, through states, when they receive or generate events. The generated events appear either on transitions or at the input or exit of states. They are noted *evt*. The received events appear on transitions. They are noted trg. Fig. 5.2 shows the state machine of a peer.

In the OPN approach, classes are represented by subnets that can be instantiated as many times as needed to describe, in a nominative manner, the objects dynamics. This instantiation is done using tokens, written in the form of n-tuples, to model class instances. According to the object-oriented concepts, the subnet encapsulates the attributes and class methods. The attributes are expressed as components of the n-tuple. As for the methods, they are specified in a flow of places, transitions and functions describing the object life cycle. Places are categorized into simple and super places. The simple places are those defined for ordinary Petri nets [21]. They include single tokens. The super places generate these tokens. Transitions are also of two types: simple and super. A simple transition models a single action. The super transition represents an internal processing described by a set of actions. Transitions can be guarded.

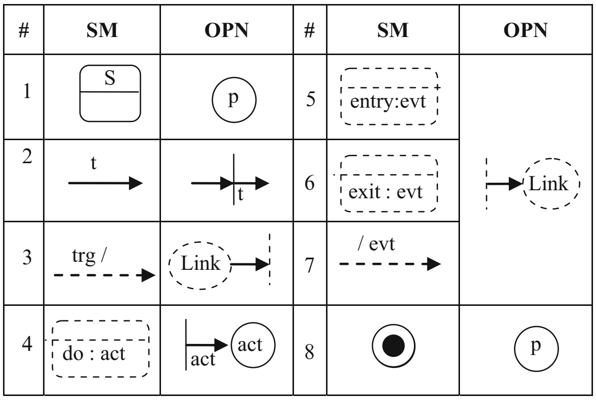


Fig. 5.3 Transformation of SM constructors into OPNs

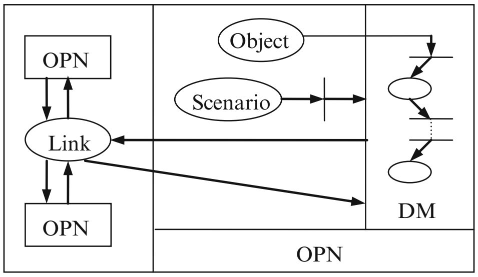


Fig. 5.4 OPNs interconnection architecture

Due to UML and OPNs suitability for the object-oriented modelling, we proposed in [8] to specify the semantics of state machines by means of OPNs. The mapping results are represented in Fig. 5.3.

Thus, each SM is derived into an object subnet called Dynamic Model or DM, see Fig. 5.4. To construct the DM, each SM state is converted to a Petri net place and each SM transition is converted to a Petri net transition related to input and output arcs. As for do activity, it is translated to a pair of transition-place connected by arcs, see Fig. 5.3. Only active objects have a behavioral model, a SM for instance. Passive objects are exchanged messages. They haven't their own behavior.

Petri nets initial marking is of two types: static and dynamic. The static marking provides the class instances and their attribute values. These instances are extracted from the object diagram to initialize the *Object* place with tokens of *object* type. The dynamic marking provides the exchanged messages among the interactive objects. These messages are extracted from the sequence diagram to initialize the *Scenario* place with tokens of *event* type.

The *DM* associated to the places *Object* and *Scenario* constitutes an Object Petri net Model that we call *OPN*. To connect the different *OPNs*, we use the *Link* place through which all the exchanged messages should pass.

Figure 5.5 shows the peer OPN derived from its state machine. The bold places *connectedPeer*, *transmittedInfo*, *receivedInfo* and *transmittedDisc* represent association ends.

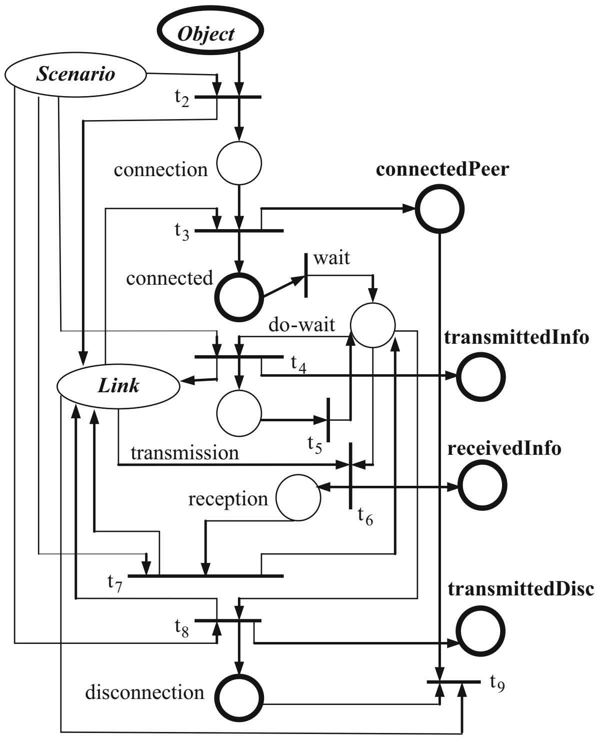


Fig. 5.5 OPN of a peer

Association End Specification

The used approach to validate OPN models is based on their consistency with the system properties transcribed in temporal logic. However, the formulation of system properties in temporal logic can be a hard task for the UML designer unfamiliar with this class of formalism. To spare him this task, we propose that he expresses the properties in a familiar language and we take care of transforming the properties in temporal logic. OCL, Object Constraint Language [26], seems to be the appropriate formalism. It is a part of the UML notation allowing the expression of constraints on models while conserving their readability.

OCL is mainly based on the use of operations on collections for specifying object invariants. Since these collections correspond to association ends, the latter must appear on Petri net specification so that the translated LTL and CTL properties (whose expression is essentially made of these constructs) can be verified. This requires the integration of the association ends onto the state machines in order to get, after their transformation, the equivalent Petri net constructs. This object flow modeling is realized by means of the link actions. But the latter are generally omitted in the behavioural diagrams. Indeed, when constructing his diagrams, the designer does not necessarily think of modeling these concepts which are rather specific to the link and association end updates. For example, for connecting a peer to the server, the connection request and connection confirmation actions are naturally and systematically modeled by the designer, but the addition of the connected peer to the association end is usually omitted from the modeling, see Figs. 5.2 and 5.5. That is why we recommend to the designer to specify the link actions on the state machine so that the OCL invariants can be verified.

UML action semantics was defined in [25] for model execution and transformation. It is a practical framework for formal descriptions. For this work, we are particularly interested in the create link, and destroy link actions. The create link action permits the addition of a new end object in the association end. The destroy link action removes an end object from the association end. These actions will be represented on the state machine as constraints of the form linkAction(associationEnd), following the event which provokes the association end update.

In Fig. 5.2, once the peer is connected (by reception of "send" okConnection) or disconnected (by reception of "send" okDisconnection), it adds or removes itself from the association end connectedPeer, using respectively, createLink(connectedPeer) or destroyLink(connectedPeer). It adds a sent or received information with createLink(transmittedInfo) or create-Link(receivedInfo), respectively.

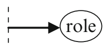
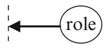
The link actions may concern an active or passive end object. The active objects interact exchanging passive objects. For example, in the peer to peer application, the *Server* and *Peer* objects are active while the *Information* object is passive.

The object-oriented approach, on which both UML and Petri nets rely, is based on modularity and encapsulation principles. To deal with modularity, a given association end should appear and be manipulated in only one state machine. In Petri nets, the association end is modeled by a place of *role* type. This place holds the name of the association end and belongs to the *DM* translating the state machine.

Furthermore, an association end regrouping active objects must be updated within the state machine of the class of these objects, in order to comply with the encapsulation concept. Indeed, since the end object is saved in the role place with its attributes, these attributes must be accessible when adding the object to or removing it from the association end. The exchanged objects are usually manipulated by the active objects and are not specified by dynamic models. So, the association end representing them could be updated in the state machine of the class that is at the opposite end. For exchanged objects, the encapsulation constraint is lifted given that the exchanged object's attributes are transmitted within the message and so, accessible by the active objects.

The create link action is semantically equivalent to a Petri net arc going from the transition with the association end update towards the place specifying the association end. The destroy link action is semantically equivalent to an arc from the association end place to the transition corresponding to the link action, see Fig. 5.6.

Fig. 5.6 Transformation of the link actions into OPNs

State machine constructs	OPN constructs
{CreateLink(role)}	
{DestroyLink(role)}	

Initialization Approach

The verification of OPNs models, derived from state machines, requires the initialization of the specification. Most of the research works [12, 29, 31] undertake this validation with an initial marking made of anonymous objects. Such marking is appropriate when one has to evaluate particularly the objects dynamics characteristic. When the interactivity feature is taken into account, the verification with anonymous objects proves to be insufficient because it inhibits many aspects of the communication. Indeed, running the verification by considering a single object as class representative may remove any meaning to inter-classes communication, especially when anonymity is on the exchanged messages.

To remedy this, we initialize the marking of OPNs models by considering objects identified by names and attribute values. Thus, the object is identified by the 2-tuple <obj, attrib> where obj is its identity and attrib, its attribute values. When getting a role through an association end, it is identified by the 3-tuple <assoc, obj, attrib> where *assoc* designates the identity of the object to which it is associated. The initialization is deduced from object and sequence diagrams.

Object and Sequence Diagrams

The object diagram [27], also called instances diagram, shows the structural links between class instances at a given time. It thus constitutes the system structural state at one a precise moment. It is composed of objects, symbolized by rectangles with two compartments. The first compartment contains the instance name concatenated to the one of the class as follows: object:Class. The state of the object may be specified in brackets. It corresponds to the object state on the state machine diagram at a given time. In the second compartment, the attributes of the object are initialized with values. The associations between objects show the links between these objects at a given time, see Fig. 5.7.

Sequence diagrams are a very attractive visual notation, widely used for modeling specific behaviors, related to the system dynamics. These behaviors are also called scenarios. They describe interactions by providing the sequence of messages exchanged between objects. Each participant in the interaction (or object) is represented by a vertical lifeline and is identified by a name appended to the one of the class as follows: object:Class. Call, send, create and destruct messages are

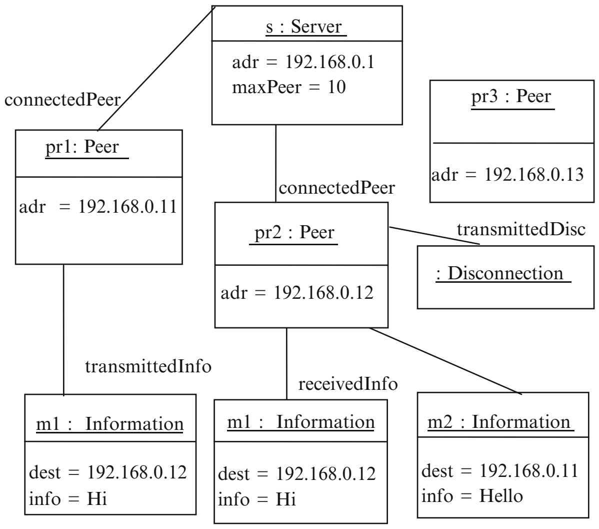


Fig. 5.7 Object diagram of a peer

respectively specified with attribute values of called operation, exchanged objects, created objects or destroyed objects as follows: "call" operation(attrib), <<send>> object: Class(attrib), <<create>> Class(attrib) or <<destroy>> Class(attrib) where $attrib = attribute_1, \ldots, attribute_n$. These messages are generated by a source object in direction of a target object. The local operations are modeled by loop arrows on the object lifeline, see Fig. 5.8.

Distribution of the Objects on the OPN

To allow an OPN simulation starting from any state of the model lifecycle, the objects (tokens) can't be put into the OPN Object place. They must be appropriately distributed into the OPN places. These places correspond to the states of the state machine from which the OPN derives. The marking of the OPN by means of objects and association ends is given by the procedure MarkObject(OD, SD, SM).

Procedure MarkObject(OD, SD, SM)

- Let OD be the object diagram, SD the sequence diagram and SM the state machine modeling the behavior of an object class ;
- For each active object obj on the OD:
  - Get the first action to be executed on the corresponding lifeline ; let act this action ;
  - Fetch, on the SM modeling the object behavior, the state(s) including act ;

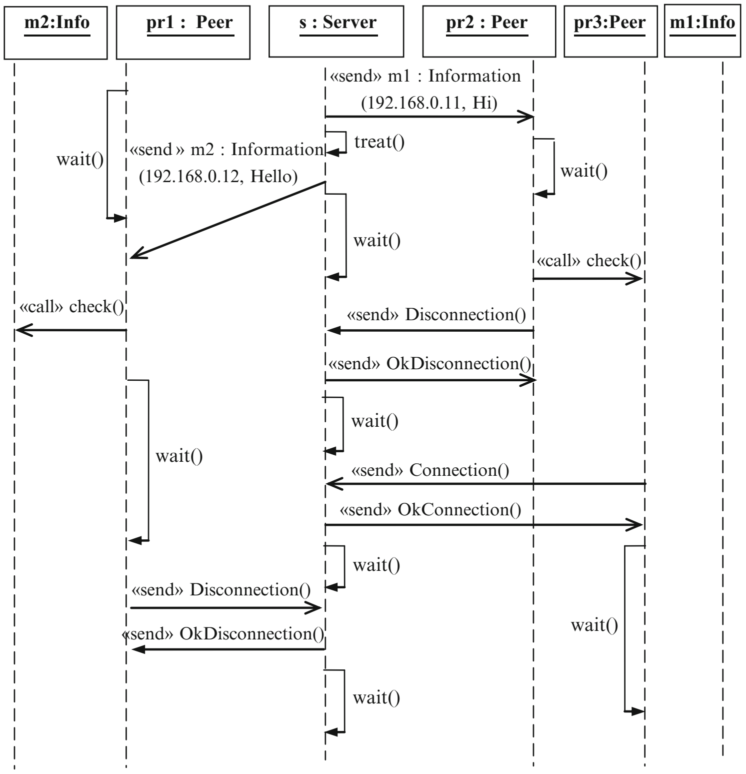


Fig. 5.8 Sequence diagram of the peer to peer system at time t

- Let s be the appropriate state ; if more than one state are found this decision is made by the designer ;
- Create a token ¡obj, attrib¿ where attrib represents the object attribute values specified on the OD ;
- Let p be the OPN place derived from the corresponding state s on the SM ;
- Put the token in the place p ;

• For each association end representing an active object specified on the OD, let asc be this association end :

- For each link specified on the OD such that $link = asc$:

- Create a token <assoc, obj, attrib>, assoc is the class to which the object obj is associated and attrib represents the object attribute values specified on the OD ;
- Let be rol an OPN place representing an association end asc ;
- Put the token in the place rol.

To illustrate this concept, we propose the object diagram of Fig. 5.7 where the server s is already connected to pr1 and pr2 peers. The peer pr3 is not yet connected. The peer pr1 is in a connected state after it has sent the message m1. The peer pr2 has received the message m1, answered it by the message m2 and then placed itself in a disconnection state.

After *MarkObject(OD,SD,SM)* has been executed, the marking of the OPN derived from the Peer state machine is given in the bold places of Fig. 5.5, as follows :

Place *Object : $pr3 : Peer$*, 192.168.0.13

Place *connected :* $< pr1 : Peer$, 192.168.0.11 $>$

Place *disconnection:* $< pr2 : Peer$, 192.168.0.12 $>$

Place *connectedPeer:* $< pr1 : Peer$, 192.168.0.11 $> +$
$< pr2 : Peer$, 192.168.0.12 $>$

We observe that the place *Object*, which usually contains the initial marking regarding objects when the used time is zero, only contains, in this case, the object pr3. We explain this arguing that pr3 has none action on its lifeline. It also presents none link on the object diagram. So it is at time zero of its life cycle. The places *connected* and *disconnected* are initialized with tokens representing objects at a time, of their life cycle, which is different from zero. As for the place *peerConnected*, it represents an association end and is initialized according to the used link on the object diagram, namely two links.

Validation of the Approach

To test the proposed approach, we built a tool whose components work as follows. We first developed a graphic interface to construct the used UML diagrams, namely state machines, object and sequence diagrams. We afterwards implemented a translator which derives OPNs from state machines. The derived OPNs were proved to be well constructed and faithful to the client requirements by means of the model checker PROD [28].

Verification by model checking as treated in PROD is based on the state space generation and the verification of safety and liveness system properties on this space. The properties may be basic, about the correctness of the model construction or specific written by the modeler to ensure the faithfulness of the system modeling. For each of these approaches, given a property, a positive or negative reply is obtained. If the property is not satisfied, it generates a trace showing a case where it is not verified.

The basic properties are verified according to two ways: the on-the-fly tester approach and the reachability graph inspection approach. The on-the-fly tester approach detects deadlock, livelock and reject states. As for the reachability inspection approach, it permits the verification of some other properties such as quasiliveness, boundedness or reinitializability. To validate the system faithfulness with the client requirements, specific properties are written by the designer in OCL, automatically translated into Linear Temporal Logic (LTL) and then, verified by PROD. Three of these properties are expressed below in a paraphrased (textual) form and then, specified as OCL invariants and translated into LTL properties. To make easier the comprehension of the properties, refer to the class diagram of the peer to peer application (Fig. 5.1).

Property 1

The number of connected peers is limited to maxPeer.

Property 1 expression in OCL

context s:Server inv : $s.connectedPeersize <= s.maxPeer$

Property 1 expression in PROD

For each server s and for each place of its *DM** write the property: # verify henceforth $(card(connectedPeer : field[0] == s_{server}) <= (placeDM * Server : field[2]))$

where:

- Field[0] designates the first component (assoc) of the *connectedPeer* tokens,
- Field[2] designates the third component ($attrib_2 = maxPeer$) of the tokens of the server *DM**.

Property 2 Only connected peers can transmit messages.

Property 2 expression in OCL

Context *s:Server* inv : $s.connectedPeer \rightarrow excludes(pr1 : Peer)$ implies $pr1.transmittedMessage \rightarrow$ isEmpty()

Property 2 expression in PROD

# verify henceforth $(connectedPeer : (field[0] == s_{server}\&\&$
$field[1] == pr1_{Peer}) == empty$ implies
$(transmittedMessage: field[0] == pr1_{Peer}) == empty)$

where :

- $Connectedpeer: field[0] == s_{server}\&\&field[1] == pr1_{Peer}$ designate the first and second components of the connectedPeer tokens,
- $Transmittedmessage$: $field[0] == pr1_{Peer}$ designates the 1st component of the *transmittedMessage* tokens.

Property 3 While a peer pr2 is connected, it receives all the information transmitted from a peer pr1.

Property 3 expression in OCL context *s:Server* inv : $s.connectedPeer \rightarrow includes(pr2 : Peer)$ and $pr1.transmittedInfo \rightarrow includes(m1 : Information)$ implies will $pr2.receivedInfo \rightarrow includes(m1 : Information)$

Property 3 expression in PROD # verify henceforth $((connectedPeer: (field[0] == s_{server}\&\&field[1] == pr2_{Peer})! = empty)\&\&(transmittedInfo : (field[0] == pr1_{Peer}\&\&field[1] == m1_{Information})! = empty)$ implies eventually $(receivedInfo : (field[0] == pr2_{Peer}\&\&field[1] == m1_{Information})! = empty))$;

where:

- $Connectedpeer: field[0] == s_{server}\&\&field[1] == pr2_{Peer}$ designate the 1st and 2nd components of the *connectedPeer* tokens,

- *Transmittedinfo* : $field[0] == pr1_{Peer}\&\&field[1] == m1_{Information}$ designate the 1st and 2nd components of the *Transmittedinfo* tokens.
- *Receivedinfo* : $field[0] == pr2_{Peer}\&\&field[1] == m1_{Information}$ designate the 1st and 2nd components of the *Receivedinfo* tokens.

Once the OPNs generated and then verified at the starting time of the system life cycle, we used object and sequence diagrams defined at specific times, different from time zero, to initialize them. This was performed using the implemented algorithm MarkObject(OD, SD, SM). The obtained markings reveal to be conformed to the object and sequence models.

Conclusion and Perspective

Many research results are published on the formalization of the UML but none so far on the initialization of the derived formal models, starting from UML diagrams set at times different from time zero. This paper proposes an approach to validate models, derived from state machines, at any time of the system life cycle. The initialization of these models is obtained from object and sequence diagrams. To locate the marks on the OPNs, the key idea focuses on the relationship between the sequence and state diagrams.

We also proposed an approach to express the association ends on the OPNs. The relevance of such an approach is to exploit all OCL capabilities to formally validate the system properties. These capabilities concern most of the OCL expressions. The only constraint of the proposed solution concerns the obligation for the user to specify the link actions on the state machine. However, this constraint is minimal compared to that of limiting OCL expressions or specifying using formal languages like temporal logics.

An interesting perspective to this work is to perform OPNs model initialization without resorting to the use of the first actions of the lifelines to locate the object states. The use of the association ends, modeled on object diagrams, is a promising research direction.

References

1. Andrade E, Macie0l P, Callou G, Nogueira B (2008) Mapping UML interaction overview diagram to time petri net for analysis and verification of embedded real-time systems with energy constraints. CIMCA 2008, Vienna
2. Andrade E, Maciel P, Callou G, Nogueira B, Araũjo C (2009) Mapping UML sequence diagram to time petri net for requirement validation of embedded real-time systems with energy constraints. SAC'2009, Hawaii, pp 377–381
3. Baresi L (2002) Some premilinary hints on formalizing UML with object petri nets. The 6th world conference on integrated design and process technology, Pasadena

4. Baresi L, Pezzè M (2001) On formalizing UML with high-level Petri Nets. Concurrent Object-Oriented Programming and Petri Nets, Advances in Petri Nets Series. LNCS. Springer, pp 276–304
5. Baresi L, Pezzè M (2005) Formal interpreters for diagram notations. ACM Trans Softw Eng Methodol 14(1):42–84
6. Baresi L, Morzenti A, Motta A, Rossi M (2011) From interaction overview diagrams to temporal logic. MODELS'10 Oslo LNCS 6637:90–104
7. Bokhari A, Poehlman WPS (2006) Translation of UML models to object coloured petri nets with a view to analysis. SEKE 2006, San Francisco, pp 568–571
8. Bouabana-Tebibel T (2007) Object dynamics formalization using object flows within UML state machines. Enterp Model Inf Syst Archit 2(1):26–39
9. Bouabana-Tebibel T (2011) Language integration for model formalization. The 12th 2011 IEEE international conference on information reuse and integration, Las Vegas
10. Bouabana-Tebibel T, Belmesk M (2007) An object-oriented approach to formally analyze the UML 2.0 activity partitions. Inf Softw Technol 49(9–10):999–1016
11. Bowles J, Andrews S, Kloul L (2010) Synthesising PEPA nets from IODs for performance analysis. WOSP/SIPEW '10, San Jose
12. Delatour J, De Lamotte F (2003) ArgoPN: A CASE tool merging UML and petri nets. The 1st international workshop on validation and verification of software for enterprise information systems, Angers
13. Fish A, Störrle H (2007) Visual qualities of the unified modeling language: deficiencies and improvements. IEEE symposium on visual languages and human-centric computing, Coeur d'Alène pp 41–49
14. Flake S (2003) UML-based specification of state-oriented real-time properties. PhD thesis, Faculty of Computer Science, Electrical Engineering and Mathematics, Paderborn University, Germany
15. Flake S, Mueller W (2004) Past- and future-oriented temporal time-bounded properties with OCL. 2nd international conferance on software engineering and formal methods, Beijing. ©IEEE Computer Society, pp 154–163
16. Guangyu Li, Yao S (2009) Research on mapping algorithm of UML sequence diagrams to object petri nets. WRI Glob Congr Intell Syst 4:285–289
17. Harel D, Maoz S (2006) Assert and negate revisited: modal semantics for UML sequence diagrams. 5th international workshop on scenarios and state machines: models, algorithms, and tools. ACM, New York, pp 13–20
18. Harel D, Kugler H, Pnueli A (2005) Synthesis revisited: generating statechart models from scenario-based requirements. In: Formal methods in software and system modeling. LNCS, vol 3393. Springer, pp 309–324
19. Holscher K, Ziemann P, Gogolla M (2006) On translating UML models into graph transformation systems. J Vis Lang Comput 17:78–105
20. Hsiung P-A, Lin S-W, Tseng C-H, Lee T-Y, Fu J-M, See W-B (2004) VERTAF: an application framework for the design and verification of embedded real-time software. IEEE Trans Softw Eng 30(10):656–674
21. Jensen K (1998) An introduction to the practical use of coloured petri nets. Lectures on Petri Nets II: Applications. LNCS, vol 1492. Springer, pp 237–292
22. Kloul L, Filipe KJ (2005) From intraction overview diagrams to PEPA nets. The work-shop on PASTA. Edinburgh
23. Knapp A, Wuttke J (2007) Model checking of UML 2.0 interactions. LNCS, vol 4364. Springer, pp 42–51
24. Kong K, Zhan K, Dong J, Xu D (2009) Specifying behavioral semantics of UML diagrams through graph transformations. J Syst Softw 82:292–306
25. Object Management Group (2001) The UML action semantics
26. Object Management Group (2003) UML 2.0 OCL specification
27. Object Management Group (2011) UML 2.4.1 superstructure specification

28. PROD 3.4 (2004) An advanced tool for efficient reachability analysis. Laboratory for Theoretical Computer Science, Helsinki University of Technology. Espoo
29. Saldana JA, Shatz SM, Hu Z (2001) Formalization of object behavior and interactions from UML models. Int J Softw Eng Knowl Eng 11(6):643–673
30. Staines TS (2008) Intuitive mapping of UML 2 activity diagrams into fundamental modeling concept petri net diagrams and colored petri nets. 15th IEEE inttenational conferance and workshop on the engineering of computer based systems, Belfast. IEEE Xplore, pp 191–200
31. Störrle H, Hausmann JH (2005) Towards a formal semantics of UML 2.0 activities. Softw Eng 64:117–128
32. Truong N, Souquiéres J (2004) Validation des propriétés d'un scénario UML/OCL à partir de sa dérivation en B. Approches Formelles dans l'Assitance au Développement de Logiciels, France