International Journal of Computer Science and Business Informatics

IJCSBI.ORG

Design of ATL Rules for Transforming UML 2 Sequence Diagrams into Petri Nets

Elkamel Merah
University of Khenchela, MISC Laboratory, University of Constantine, Algeria

Nabil Messaoudi
University of Khenchela, MISC Laboratory University, of Constantine, Algeria

Dalal Bardou
University of Khenchela, Algeria

Allaoua Chaoui
MISC Laboratory, University of Constantine, Algeria

ABSTRACT
UML 2 sequence diagrams are a well-known graphical language and are widely used to specify the dynamic behaviors of transaction-oriented systems. However, sequence diagrams are expressed in a semi-formal modeling language and need a well-defined formal semantic base for their notations. This formalization enables analysis and verification tasks. Many efforts have been made to transform sequence diagrams into formal representations including Petri Nets. Petri Nets are a mathematical tool allowing formal specification of the system dynamics and they are commonly used in Model Checking. In this paper, we present a transformation approach that consists of a source metamodel for UML 2 sequence diagrams, a target metamodel for Petri Nets and transformation rules. This approach has been implemented using Atlas Transformation Language (ATL). A Cellular Phone System is considered, as a case study.

Keywords
UML 2, Sequence diagrams, Petri Nets, Model checking, Model transformation, Metamodeling, Transformation rules, ATL.

1. INTRODUCTION

The Unified Modeling Language (UML) [2] is a general-purpose graphical object-oriented modeling language that is designed to visualize, specify, construct and document software systems in both structural and behavioral aspects. UML is intended to be a common way of capturing and expressing relationships and behaviors in a notation that is easy to learn and efficient to write [17].

In 1997, UML [2] was accepted by the Object Management Group (OMG) [13][17]. Since then, UML has gone several revisions and refinements leading up to the current UML, revision 2 [13][17]. This revision represents the cleanest, most compact version of UML. Today, UML is widely accepted by the software engineering community as a standard in industry and research.

The UML 2 provides several categories of diagrams to specify different aspects of the system, like structural or behavioral aspect. For behavioral-intensive systems, the dynamic behavior is the most critical aspect to take into account.

Sequence Diagrams (SDs) - which are considered in this paper - be- long to the behavioral diagrams like communication diagrams. They are collectively known as interaction diagrams. The communication diagrams are used to understand and document the interactions between the objects and also in order to show how the classes are working together to achieve a goal [11]. Sequence diagrams emphasize the type and order of messages passed between elements during execution [17]. We selected SDs from UML 2 interactions diagrams since they are the most common type of interaction diagrams and are very intuitive to new users of UML [17].

UML models of the interaction category are generally transformed for verification and validation purposes. This is because dynamic models, such as SDs, lack sufficient formal semantics [9]. Moreover, UML was created as a semi-formal modeling language it does not include a formal semantics [4]. This limitation makes rigorous analysis difficult, which leads to an ambiguous model and problems with modeling the process concurrency, synchronization, and mutual exclusion [21]. On the other hand, one of the most important problems of designing phase in software engineering is to verify all designed things before going to the implementation phase because starting the implementation phase before verifying design phase is a big risk in big projects [11].

Thus, production of the new technologies for verification and validation of UML models seem very crucial and converting UML to some mathematical models, in order to formalize and validate them can be a very important task. Many researchers have been performed in order to only transform the UML models into a formal model [11]. In our approach, the formal model is Petri Nets (PNs) [12][18]. Petri Nets can model, among others like automata, the behavior of systems having concurrency. Since PNs are a formal model and they have a mathematical representation with a well-defined syntax and semantics, they do not carry any ambiguity and thus, are able to be validated, verified and simulated.

The suggested approach is mainly based on the technique of metamodel transformations [5]. Such approach consists in defining the source metamodel of sequence diagrams, defining the target metamodel of Petri Nets, and defining the transformation rules. Our transformation contributes to the on-going attempt to develop a formal semantics of UML [13] based on model transformations [5]. On the basis of this transformation it is possible to accomplish verification of the dynamic model of the real system. All these reasons motivate the work to map or to transform UML 2 sequence diagrams to Petri Nets. To achieve this goal, this paper proposes a set of rules for this transformation.

The rest of this paper is organized as follows. In section 2, we discuss related work. In section 3, we briefly review the features of UML 2 interactions and sequence diagrams and we briefly introduce Petri Nets. Section 3 also describes both source and target metamodels suited to the transformation. We then show in section 4 how we translate a sequence diagram into behaviorally equivalent Petri Net (PN). In section 5 is presented the application of the proposed transformation rules with a Cellular Phone System. Section 6 presents the implementation of the system design transformation process. We finally conclude our work in section 7 with some remarks and future work.

2. RELATED WORKS

Many research works have been done on model transformations and especially to transform sequence diagrams into Petri Nets in order to perform formal verification. UML sequence diagrams have been very considered and many works propose a rule-based approach to automatically trans- late sequence diagrams into Petri Nets.

Kessentini [9] describes an automated SDs to colored Petri Nets transformation method, which finds the combination of transformation fragments that best covers the SD model, using heuristic search in a base of examples. To achieve his goal, he combines two algorithms for global and local search, namely Particle Swarm Optimization (PSO) and Simulated Annealing (SA). Ait-Oubelli [15] uses graph transformation to transform SDs to Promela code. Ribeiro [19] proposed a set of rules that allow software engineers to transform UML 2.0 sequence diagrams into a Colored Petri Net. He also used graph transformation to specify transformation rules. Chaoui [3] proposed an approach to translate SDs models to their equivalent ECATNets models. The resulting models can be subjected to various Petri Net analysis techniques. His approach TNets models are graphs.

In another work, we have proposed a similar approach [10] but deals with UML 2.0 communication diagrams.

This paper deals with transforming UML 2 sequence diagrams into Petri Nets models for analysis and verification purposes by using some transformation rules expressed in the ATL language. Our work is a step forward in a project that is exploring means to define a semantics for UML 2 communication diagrams.

3. THE BASIC METAMODELS

3.1 UML 2 Diagrams For Interaction

UML 2 divides diagrams into two categories: structural modeling diagrams and behavioral modeling diagrams:

• Structural diagrams illustrate the static features of a model. Static features include classes, objects, interfaces and physical components. In addition, they are used to model the relationships and dependencies between elements. Structural diagrams include Class diagram, Object diagram, and some others.

• Behavioral diagrams describe how the modeled resources in the structural diagrams interact and how they execute each other capabilities. The behavioral diagram puts the resources in motion, in contrast to the structural view, which provides a static definition of the resources [16]. Behavioral diagrams include the Interaction diagrams, Use Case diagram, Activity diagram, State Machine diagram and others.

Interaction diagrams [17] are defined by UML 2 to emphasize the communication between objects, not the data manipulation associated with that communication. Interaction diagrams focus on specific messages between objects and how these messages come together to realize functionality [17]. An interaction can be displayed in several different kinds of diagrams: Sequence Diagrams, Communication Diagrams, Interaction Overview Diagrams, and Timing Diagrams.

– Sequence diagrams are one of the kinds of interaction diagrams that emphasize the type and order of messages passed between elements during execution [17].

– Communication diagrams are one of the kinds of interaction diagrams that focuses on the elements involved in a particular behavior and what messages they pass back and forth [17]. Communication diagrams emphasize the objects involved more than the order and type of the messages exchanged [17].

– Interaction overview diagrams are simplified versions of activity diagrams [17]. Instead of emphasizing the activity at each step, interaction overview diagrams emphasize which element or elements are involved in performing that activity [17].

– Timing diagrams are designed to specify the time constraints on messages sent and received in the course of an interaction. They are often used to model real-time systems such as satellite communication or hardware handshaking [17].

Both sequence and communication diagrams concentrate on the presentation of dynamic aspects of a software system, each from a different perspective. Sequence diagrams stress time ordering while communication diagrams focus on organization. Despite their different emphases, they share a common set of features. Booch, Rumbaugh, and Jacobson [2] claim that they are semantically equivalent since they are both derived from the same sub-model of the UML metamodel, which gives a systematic description of the syntax and semantics of the UML. In this work, we concentrate on sequence diagrams.

3.1.1 Sequence Diagrams

Sequence Diagrams (SDs) and Communication Diagrams (CDs) are two views of the same scenario where SD gives the temporal view of a scenario and CD gives the structural one. SDs record the same information as CDs and, hence, scenarios. They just provide a different view one that focuses on the structural view of the object interactions, rather than the temporal view. The communication is implicit in a SD, rather than explicitly represented as in a CD. Some tools even generate SDs from CDs (or vice versa).

3.1.2 Sequence Diagrams Metamodel

Sequence diagram expresses interactions between objects by exchanging messages. We provide UML 2 sequence diagram a metamodel, which graphically displays the abstract syntax in terms of class diagram. The metamodel complies with the interaction metamodel provided by OMG [13], whereas showing only the essential syntax constructs of a sequence diagram, to facilitate the mapping to the Petri Nets. In the metamodel, the syntax elements are represented as classes, shown as boxes, and relations elements are represented as associations, shown as lines among classes in terms of class diagram. A hollow diamond on an association represents aggregation relationship (has-a), while a filled diamond represents a composition relationship (part-of). A triangle on an association represents a generalization between a superclass and its subclass. The numbers attached to an association are called multiplicities, which describe how many objects may exist in the association. A star denotes zero or more. If no multiplicity is present, a one-to-one relationship is implied [20].

In this work, we proposed a sequence diagram metamodel, it is inspired from the OMG [13] metamodel. It describes all the concepts and the

relations existed between them. Figure 1 shows our simplified metamodel for UML 2 sequence diagrams. The important concepts in an interaction are life lines, messages, and combined fragments.

- Description of the metamodel:
  - The class Interaction: It is the root, which represents an interaction. Each interaction has a name (attribute name of type String). An interaction consists of a set of life lines and a set of messages.
  - The class LifeLine: LifeLine represents the operations executed by an object. Each life line has a set of incoming and outgoing messages. It can be covered by interaction operands.

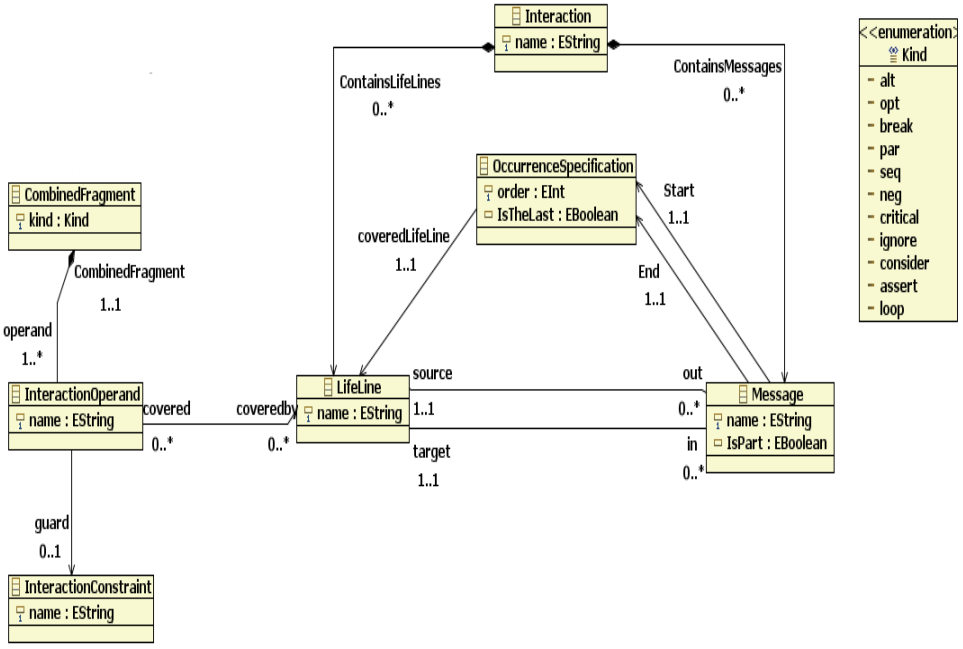

Figure 1. A simplified metamodel for UML 2 sequence diagrams

- The class Message: A message defines a particular communication between two objects. Each message has a name, the attribute IsPart of type Boolean is used for the transformation of the operator Alt. Message consists of Send and Receive action, which are placed on two different Occurrences Start and End.
- The class OccurrenceSpecification: It describes the scheduling of messages, through order attribute of type Int. The attribute IsTheLast of type Boolean is used to differentiate between the last message and the others.

- The class CombinedFragment: It consists of a set of operands of type IntercationOperand. Each CombinedFragment has a kind; it takes a value among the enumeration Kind values.
- The class InteractionOperand: It represents an operand of an operator, it has a name and it can have a constraint of type InteractionConstraint. An interaction operand covers by a set of life lines.
- The class InteractionConstraint: It has an attribute name of type String that represents the value of the constraint.

3.2 Petri Nets Metamodel

Petri Nets are a graphical and mathematical representation of discrete distributed systems. They are also known as Place/Transition nets or P/T nets. Petri Nets consist of places, transitions and directed arcs to connect them. There are two sorts of arcs connecting place to transition or transition to place.

A Petri Net is a 4-tuple PN = (P, T, Pre, Post) where:

1. P is a finite set of places,
2. T is a finite set of transitions,
3. Pre: P*T —> N is the application of previous places,
4. Post: P*T —> N is the application of following places.

Figure 2 shows a metamodel for Petri Nets.

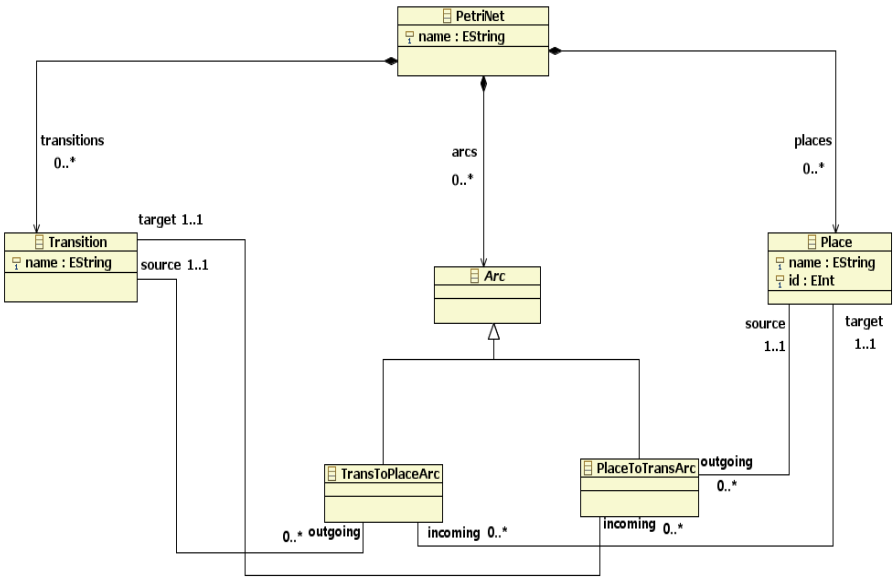

Figure 2. Petri Nets metamodel

- Description of the metamodel:
  - The class PetriNet: It's the root which represents a Petri Net, it has one attribute "name" of type String, it takes the name of the Petri Net.
  - The class Place has two attributes:
    - Name: of type String, it represents the content of the place.
    - Id: of type Int, it used for scheduling the set of the places.

Each place has a set of outgoing PlaceToTransArc, and a set of incoming TransToPlaceArc.

- The class Transition: It has one attribute"name" of type String. It represents the action executed by the transition (Send or Receive action). Each transition has a set of outgoing TransToPlaceArc, and a set of incoming PlaceToTransArc.
- Arcs are "PlaceToTransArc" or "TransToPlaceArc". The class Arc is an abstract class, it's only used for inheritance. Both of PlaceToTransArc and TransToPlaceArc inherit frome class Arc.
- Each PlaceToTransArc has as a source a place, and as a target a transition.
- Each TransToPlaceArc has as a source a transition, and as a target a place.

4. TRANSFORMATION APPROACH

4.1 The Transformation Process

To make easier the rules' specification of the transformation, our efforts address the transformation at the metamodel level of UML 2. This also allows the mapping between the concepts of both metamodels source and target. The metamodeling based transformation approach for transforming UML 2 sequence diagrams into Petri Nets is shown in Figure 3. Sequence diagrams are assumed to be syntactically and static semantically correct. The transformation process is achieved by the application of rules. A transformation rule consists in transforming a concept outlined in the source metamodel to a corresponding concept in the target meta- model.

4.2 Transformation Rules

In the following, we define the rules for transforming sequence diagrams into Petri Nets. The transformation rules describe the interactions that exist between classes of the "sequence diagrams" metamodel and " Petri Nets" metamodel. These rules consist essentially of:

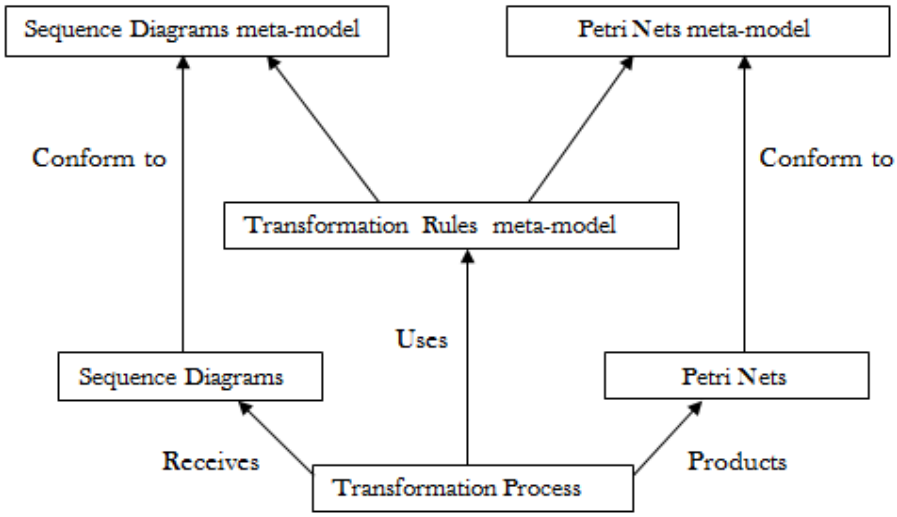

Figure 3. Overview of the Sequence Diagrams to Petri Nets ATL transformation

- Basic Interaction Transformation Rules
  - Rule1: The name of Petri Net is the name of the Interaction.
  - Rule2: Each Message is transformed into two sub-Petri nets (Figure 4). Each sub-Petri net describes the behavior of an object (the status of the object before and after Send (Receive action). These sub-Petri nets are connected with a place labeled with the message name.
- Alt Transformation Rules

Alt is transformed as shown in Figure 5.

- Rule1: The role of this rule is the verification of the kind and the number of operands. From the class CombinedFragment, places and transitions correspond to Alt transformation are initialized.
- Rule2: It's the same rule (Rule2) of Basic Interaction Transformation, the only difference that we handle all cases to connect operand begin transition with the places correspond to first send and receive messages, and places correspond to last send and receive message with operand end transition (for example, Altcase1 below).

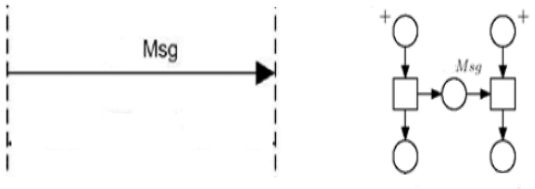

Figure 4. A message transformation

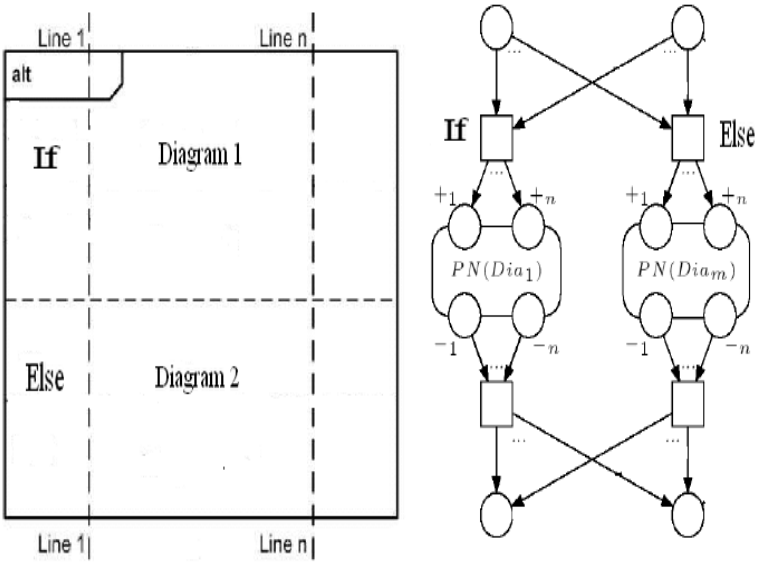

Figure 5. Alt transformation

- Parallel Transformation Rules

Parallel is transformed as shown in Figure 6.

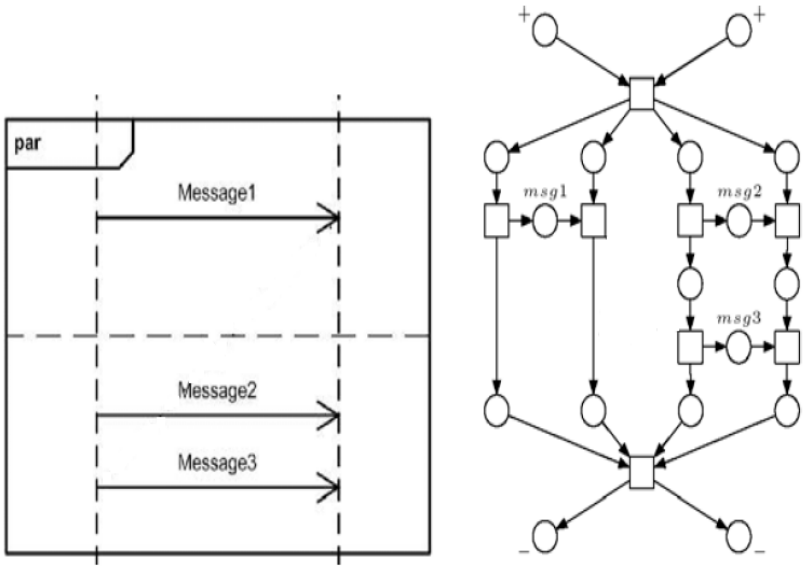

Figure 6. Parallel transformation

- Rule1: The role of this rule is the verification of the kind, the number of operands does not matter. From the class CombinedFragment, places and transitions correspond to Parallel transformation are initialized.
- Rule2: It's the same rule (Rule2) of Basic Interaction Transformation, the only difference that we handle all cases to connect Parallel begin transition with the places correspond to first send and receive messages , and places correspond to last send and receive message with Parallel end transition.

Basic Interaction Transformation Rules

```
rule Interaction2PetriNet{
from
s: SequenceDiagram!Interaction —- It produces Petri Net's name—-
to
p: PetriNet!PetriNet (name <- s.name)
}
rule Interaction{ from s:SequenceDiagram!Message to
l:PetriNet!Place( — It produces the initial send place—-
name<-'Send'+ s.name +'.Before ='+ s.SourceLifeLineName +'.Begin', id <-
s.MessageSendOrder),
n:PetriNet!Transition( —-It produces the send transition—-
name<-'Send '+ s.name +'('+ s.SourceLifeLineName+ ','+
s.TargetLifeLineName+ ')'),
r: PetriNet!Place ( ———It produces the final send place———- name<-
s.SourceLifeLineName +':Send'+ s.name +'.After', id <-
s.MessageSendOrder+1),
m:PetriNet!Place( ———-It produces the middle place———
name<- s.name),
t:PetriNet!Place ( ———-It produces the initial receive place——
name<-'Receive'+ s.name +'.Before ='+ s.TargetLifeLineName +'.Begin',
id<-s.MessageReceiveOrder),
p:PetriNet!Transition( ———-It produces the receive transition——
name<-'Receive '+ s.name +'('+ s.SourceLifeLineName+ ','+
s.TargetLifeLineName+ ')'),
d:PetriNet!Place ( ———-It produces the final receive place—— name<-
s.TargetLifeLineName+':Receive'+ s.name+'.After ', id<-
s.MessageReceiveOrder+1),
isp-st :PetriNet!PlaceToTransArc( ———-It produces for each arc their source
and target nodes, the same thing for the rest below——
source <-l,
target <-n),
...
st-mp:PetriNet!TransToPlaceArc( source<-n,target<-m)
}
```

Alt Transformation Rules

```
rule Alt{
from
c:SequenceDiagram!CombinedFragment (c.IsAlt() and c.Has2Operand()) ——
–It checks if the combined fragment is named Alt and if it has two operands———
to
pl:PetriNet!Place ( ——It produces the first operand begin place——
name<-'Alt part one:Begin'),
sl:PetriNet!Place ( ———It produces the second operand begin place——
name<-'Alt part two:Begin'),
rl:PetriNet!Place( ——It produces the first operand end place——
name<-'Alt part one:End'),
tl:PetriNet!Place ( -It produces the second operand begin place-
name<-'Alt part two:end'),
nl:PetriNet!Transition ( -It produces the first operand transition contains the
operand's name-
name<- 'ConditionOne :' + c.getFirstOperandName),
bl:PetriNet!Transition ( -It produces the second operand transition contains the
operand's name-
name<- 'ConditionTwo :'+ c.getSecondOperandName),
al:PetriNet!Transition( -It produces the first operand transition end contains the
operand's end-
name<-'ConditionOne:'+ c.getFirstOperandName +'.End '),
dl:PetriNet!Transition( -It produces the second operand transition end con-
tains the operand's end- name<-'ConditionTwo:'+ c.getSecondOperandName
+'.End '),
  ——————————The arcs' source and target nodes as Basic Interaction
Transformation——
}
```

Alt Transformation Rules

```
rule AltCase1{
from
s:SequenceDiagram!Message(s.FirstSendMessage() and
s.FirstReceiveMessage() and s.IsPartOne() and s.IsTheLastSend() and
s.IsTheLastReceive()) ——It checks which part is the message and its extremities to
attach it with the right arcs ————
to
l:PetriNet!Place( name<-'Send'+ s.name +'.Before ='+
s.SourceLifeLineName +'.Begin', id ¡-s.MessageSendOrder),
n:PetriNet!Transition(
name¡-'Send '+ s.name +'('+ s.SourceLifeLineName+ ','+
s.TargetLifeLineName+ ')'),
r:PetriNet!Place (name¡-'Send'+ s.name +'.After ='+ s.SourceLifeLineName
+'.End',
id ¡- s.MessageSendOrder+1), m:PetriNet!Place(name¡-s.name),
t:PetriNet!Place (
name¡-'Receive'+ s.name +'.Before ='+ s.TargetLifeLineName +'.Begin', id¡-
s.MessageReceiveOrder),
p:PetriNet!Transition(
name¡-'Receive '+ s.name +'('+ s.SourceLifeLineName+ ','+
s.TargetLifeLineName+ ')'), d:PetriNet!Place (
name¡-'Receive'+ s.name +'.After ='+ s.SourceLifeLineName +'.End' , id¡-
s.MessageReceiveOrder+1),
isp-st :PetriNet!PlaceToTransArc(source <-l,target <-n),... fsp1-
ft:PetriNet!PlaceToTransArc(
source<-r,
target<-thisModule.resolveTemp(thisModule.root,'al')), frp1-
ft:PetriNet!PlaceToTransArc(
source<-d,target<-thisModule.resolveTemp(thisModule.root,'al'))
}
```

Parallel Transformation Rules

```
rule Parallel{
from
c:SequenceDiagram!CombinedFragment (c.IsParallel())
to
pl:PetriNet!Place (
name<-'Operator Parallel'), pl2:PetriNet!Place (
name<-'Operator Parallel'), nl:PetriNet!Transition (
name<-'Operator Parallel Begin'), tl:PetriNet!Place (
name<-'Operator Parallel'), tl2:PetriNet!Place (
name<-'Operator Parallel'), kl:PetriNet!Transition (
name<-'Operator Parallel End'), pl-nl:PetriNet!PlaceToTransArc( source <-
pl,
target <-nl),
pl2-nl:PetriNet!PlaceToTransArc(
source <-pl2, target <-nl),
tl-kl:PetriNet!TransToPlaceArc(source <-kl,target <-tl),
pl2-nll:PetriNet!TransToPlaceArc(source <-kl,target <-tl2)
}



rule ParallelCase1 {
from
s:SequenceDiagram!Message(s.FirstSendMessage() and
s.FirstReceiveMessage() and s.IsTheLastSend() and s.IsTheLastReceive())
c:SequenceDiagram!CombinedFragment (c.IsParallel())
to l:PetriNet!Place(
name<-'Send'+ s.name +'.Before ='+ s.SourceLifeLineName +'.Begin',id
<-s.MessageSendOrder),n:PetriNet!Transition(
name<-'Send '+ s.name +'('+ s.SourceLifeLineName+ ','+
s.TargetLifeLineName+ ')'), r:PetriNet!Place (
name<-'Send'+ s.name +'.After ='+ s.SourceLifeLineName +'.End',id <-
s.MessageSendOrder+1),
m:PetriNet!Place(name<-s.name), t:PetriNet!Place (
name<-'Receive'+ s.name +'.Before ='+ s.TargetLifeLineName
+'.Begin',id<-s.MessageReceiveOrder), p:PetriNet!Transition(
name<-'Receive '+ s.name +'('+ s.SourceLifeLineName+ ','+
s.TargetLifeLineName+ ')'), d:PetriNet!Place (
name<-'Receive'+ s.name +'.After ='+ s.SourceLifeLineName +'.End'
,id<-s.MessageReceiveOrder+1),
tl-kl:PetriNet!TransToPlaceArc(source <-kl,target <-tl),
pl2-nll:PetriNet!TransToPlaceArc(source <-kl,target <-tl2)... d-
kl:PetriNet!PlaceToTransArc(source<-d,
target<-thisModule.resolveTemp(thisModule.root,'kl'))
}
```

5. A CASE STUDY: A PHONE SYSTEM

To validate the proposed transformation, we choose a Phone System as a case study. Sequence diagram shown in the Figure 7, illustrates a basic interaction between three objects Caller, Phone and Receiver. The use case of this interaction is carried out as follows:

- Caller lifts the Phone.
- Dial-tone is heard by the Caller.
- Caller composes the number.
- Caller is connected to the network (Connect tone).
- Ring-tone is heard by the Caller (Receiver is not busy).
- Receiver's phone rings.
- Receiver answers the Caller.
- Caller is talking to the Receiver.
- Receiver is talking to the Caller.
- Disconnexion operation.
- Caller hangs up.

This simple procedure is depicted in the sequence diagram in Figure 7, while Figure 8 shows a possible abstract syntax of the same diagram according to the metamodel we have defined above. The Phone System model after applying the steps of the transformation is seen in Figure 9. We have now reached a Petri Net corresponding to the Phone System.

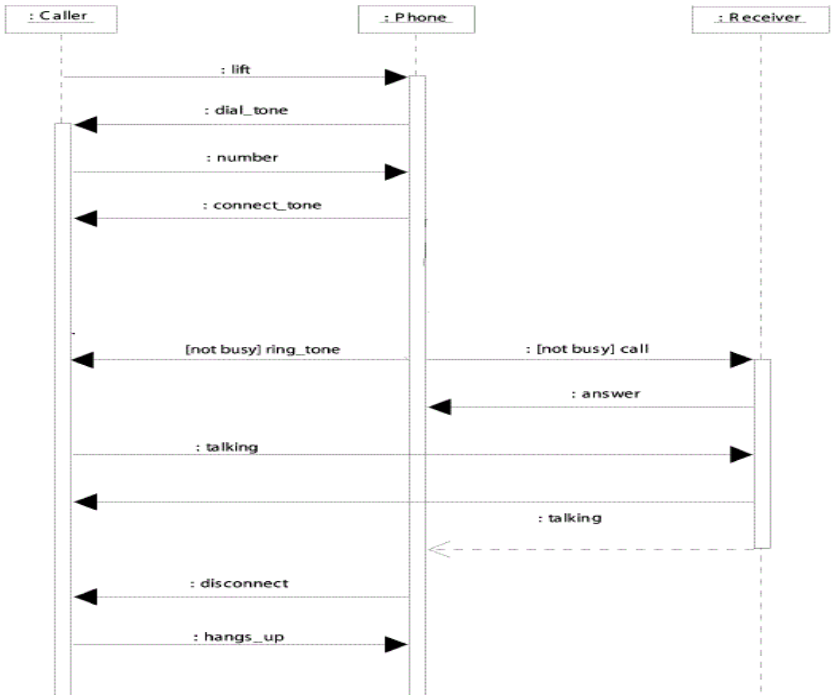

Figure 7. Sequence diagram of the phone system

6. IMPLEMENTATION

We have chosen Atlas Transformation Language (ATL) [8][7] under the Eclipse development platform [6] to express the transformation rules. ATL is a model transformation language that contains a mixture of declarative and imperative constructs. ATL is accompanied by a set of tools built on top of the Eclipse platform. According to the adopted transformation process, the implementation of this process requires the following steps:

1. The representation of the source metamodel described in UML2-sequence diagram in Ecore Diagram Tool which generates An Ecore file named Sequence Diagram.ecore described in XMI language [14].
2. The representation of the target metamodel described in Petri Nets in Ecore Diagram Tool which generates an Ecore file named PetriNet.ecore described in XMI language.
3. The representation of a model instance, i.e. a sequence diagram, of the source metamodel in Ecore file.
4. Applying the rules of model transformation specified in ATL language to the source model. This process generates an XMI file containing a Petri Net describing formally the behavior of the source sequence diagram.

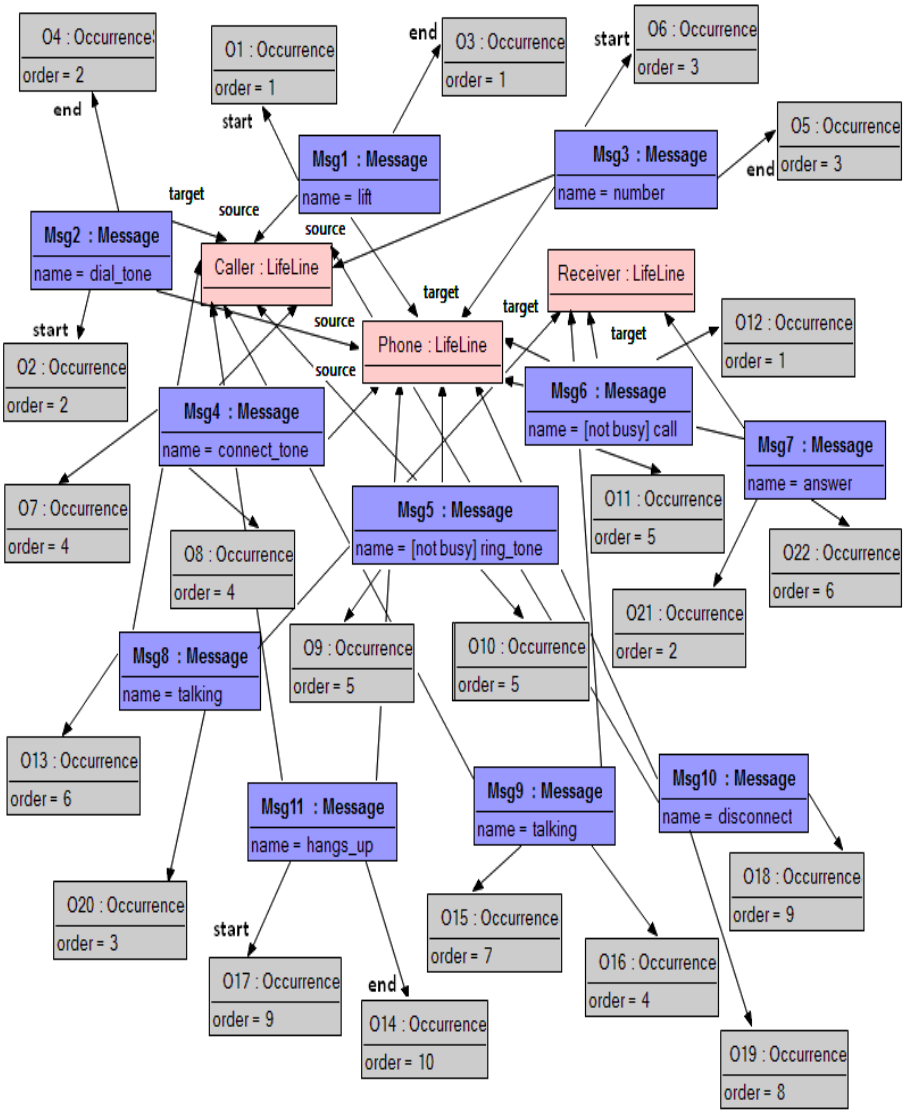

Figure 8. Sequence diagram for the phone system in abstract syntax

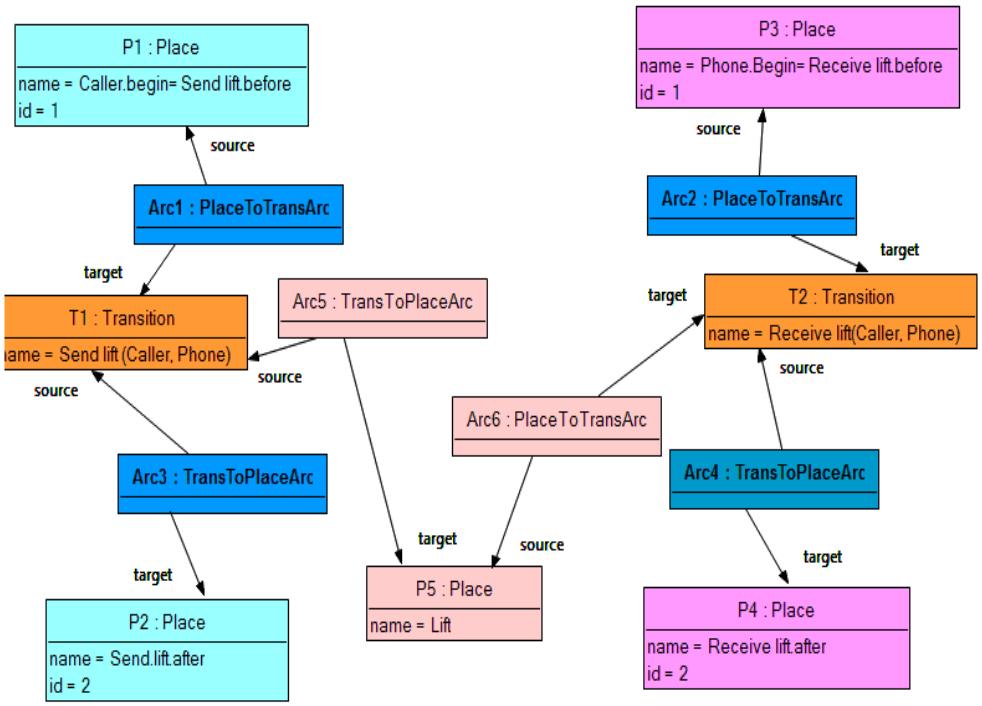

Figure 9. An extract f r o m Petri Net for the phone system in abstract syntax

7. CONCLUSION AND PERSPECTIVES

In this paper, we proposed a transformation from UML 2 sequence diagrams into Petri Nets. A set of rules was defined to govern the transformation p r o c e s s . On the basis of this transformation it is possible to accomplish verification of the dynamic model of the real system expressed by a sequence diagram. Our approach w a s implemented u s i n g the ATL language. A Phone System case study w a s used to illustrate the transformation technique. This work still in progress so we plan to complete it further. First, one direction f o r future w o r k can be to extend t h i s transformation to other operators such as ignore and loop. Second, we need to better t u n e the rules, to realize if they can be automated [1]. Third, i s to generate Java code automatically from UML 2 sequence diagrams [22].

REFERENCES

[1] W. Alouini, O. Guedhami, S. Hammoudi, M.Gammoudi, and D. Lopes. Semiautomatic Generation of Transformation Rules in Model Driven Engineering: The Challenge and First Steps. International Journal of Software Engineering and Its Applications (IJSEA), 5(1), (2011).

[2] G. Booch, J. Rumbaugh, and I. Jacobson. Unified Modeling Language, Version 1.0. Rational Software Corporation, (2010).

[3] A. Chaoui, R. Elmansouri, W. Saadi, and E. Kerkouche. From UML Sequence Diagrams to ECATNets: a Graph Transformation based Approach for modelling and analysis. In proceedings of The 4th International Conference on Information Technology ICIT 2009, June 3rd, (2009).

[4] H.Y. Chen, C. Li, and T.H. Tse. Transforming of UML Interaction Diagrams into Contract Specifications for Object-Oriented Testing. In Proceedings of the 2007 IEEE International Conference on Systems, Man, and Cybernetics, (2007).

[5] K. Czarnecki and S. Helsen. Classification of Model Transformation Approaches. In OOSPLA'03, Workshop on Generative Techniques in the Context of Model-Driven Architecture. Anaheim, USA, (2007).

[6] Eclipse Official Site: http::/www.eclipse.org.

[7] F. Jouault, F. Allilaire, J. Bézivin, I. Kurtev, and P. Valduriez. ATL: a QVT-like Transformation Language. In Proceedings OOPSLA'06 Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications, (2006).

[8] F. Jouault and I. Kurtev. Transforming Models with ATL. In J.M. Bruel, editor, MODELS Workshop, LNCS3844. Montego Bay, Jamaica, (2005).

[9] M. Kessentini, A. Bouchoucha, H. Sahraoui, and M. Boukadoum. Example-Based Sequence Diagrams to Colored Petri Nets Transformation Using Heuristic Search. In T. Kuhne et al. (Eds): ECMFA 2010, LNCS 6138. Springer-Verlag Berlin Heidelberg, (2010).

[10] E. Merah, N. Messaoudi, H. Saidi, and A. Chaoui. Design of ATL Rules for Transforming UML 2 Communication Diagrams into Büchi Automata. In International Journal of Software Engineering and Its Applications Vol.7, No.2, March, (2013).

[11] H. Motameni and T. Ghassempouri. Transforming Fuzzy Communication Diagram to Fuzzy Petri Net. American Journal of Scientific Research, 16, (2011).

[12] T. Murata. Petri nets: Properties, Analysis and Applications. In Proceedings of the IEEE, volume 77, (1989).

[13] Object Management Group. OMG Unified Modeling Language (OMG UML), Superstructure, V2.1.2. (2007).

[14] Object Management Group XMI Specification. http://www.omg/spec/xmi/2.4.1. (2011).

[15] M. Ait Oubelli, N. Younsi, A. Amirat, and A. Menasria. From UML 2.0 Sequence Diagrams to PROMELA code by Graph Transformation using AToM3. In CIIA, volume 825 of CEUR Workshop Proceedings, CEUR-WS.org, (2011).

[16] T. Pender. UML Bible. John Wiley & Sons, (2003).

[17] D. Pilone and N. Pitman. UML 2.0 in a Nutshell. O'Reilly Publisher, (2005).

[18] W. Reisig. Petri nets - An Introduction. Springer, (1985).

[19] O.R. Ribeiro and J.M. Fern. Some Rules to Transform Sequence Diagrams into Coloured Petri Nets. In In 7th Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools, (2006).

[20] H. Shen, A. Virani, and J. Niu. Formalize UML 2 Sequence Diagrams. High Assurance Systems Engineering Symposium, 2008. HASE 2008. 11th IEEE., (2008).

[21] I. Trickovic. Transformation of the State Diagram of the Unified Modeling Language into a Petri Nets Model. NOVI SAD J. MATH, 28(3), (1998).

[22] M. Usman and A. Nadeem. Automatic Generation of Java Code from UML Diagrams using UJECTOR. International Journal of Software Engineering and Its Applications (IJSEA), 3(2), (2009).