Formalization Method of the UML Statechart by Transformation Toward Petri Nets

Thierry Noulamo, Emmanuel Tanyi, Marcellin Nkenlifack, Jean-Pierre Lienou, and Alain Djimeli

Abstract—**Several studies have focused on the formalization of semi-formal methods, by derivation toward formal methods such as the formalization of UML diagrams by transformation toward Petri Nets, in order to take advantage of evidence tools to perform model-checking or code generation. The objective of this work is to propose a rigorous approach to build a UML statechart and propose and automatic method to transform it toward Time Colored Petri Nets. Our approach is to produce patterns of UML statechart, composition rules of these patterns, which will be used in the development process. The resulting diagram is automatically transformed into Time Colored Petri Nets by an algorithm that implements our derivation rules. For the validation of our model, we apply our proposal to the modelling of a machine for hot drinks.**

Index Terms—**statechart patterns diagram, temporal colored Petri nets, Formalization of models, specification of information systems.**

I. Introduction

The use of formal methods, qualified as low level approach such as automata, Petri Nets or timed automata networks, to describe the dynamic of systems, leads us to take advantage of existing verification, validation and code generation tools[36][37][38].

The Petri Nets had been used for a long time as a base model to represent behaviors of the concurrent systems, with a particular attention to the problems of asynchronous parallelism of distributed systems.

Since their introduction in [34], Petri Nets (PN) have been widely used for the specification and verification of systems [36][37][38] like communication protocols or real-time systems. However, their formalisms does not offer much flexibility to have a more accessible expressivity for the system designer[8][19]. Using a high-level approach, allows directly to express the concepts handled by the user. UML (Unified Modeling Language) is the standard notation that has been imposed by itself for the modeling of computer systems, according to the object approach. It allows designers to describe different aspects of complex systems, by using its different types of diagrams. However, UML suffers from an incessant criticism, due to its lack of formal semantics [6], which is a problem when we want to ensure of the consistency of the systems described using one or more of its diagrams and generally addressing the verification steps that follow the system modeling.

Manuscript received June 28, 2017; revised July 13, 2018.

M. Noulamo is with the Department of Computer Engineering,University of Dschang, Cameroon, e-mail: thierry.noulamo@gmail.com

M. Tanyi is with the Department of Electrical Engineering, University of Buea, Cameroon, e-mail: emmantanyi@yahoo.com

M. Nkenlifack is with the Department of Computer Engineering, University of Dschang, Cameroon, e-mail:macellin.nkenlifack@gmail.com

M. Lienou is with the Department of Computer Engineering,University of Bamenda, Cameroon, e-mail: lienou@gmail.com

M. Djimeli is with the Department of Network Telecomunication Engineering,University of Dschang, Cameroon, e-mail: adjimeli@gmail.com

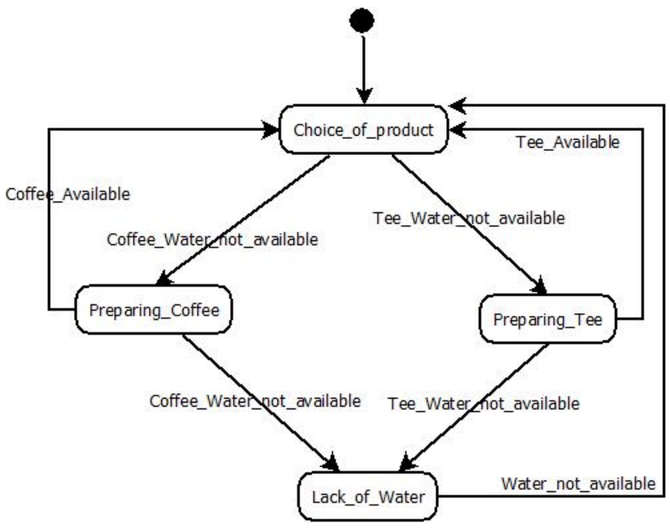


Fig. 1. Example of $UMLstatechart$ having multi-inputs and multi-outputs

The formalization of UML is now the subject of numerous studies, the main one is from the UML Precise Group [29]. The UML statechart, widely used for the description of the dynamic of the system in the objects approach, also presents weaknesses. Figure 1 is a syntactically correct example of UML statecharts, that models our case study.

In this diagram, the state ”Choice_of_product” has 4 inputs and two outputs. The states ”Preparing_Coffee” and ”Preparing_Tee” have one input and two outputs. The state ”Lack_of_Water” has two inputs and one output. This freedom available to the designer in terms of number of inputs or outputs that can have a state and the lack of composing rules for the UML statecharts, leads to an UML statechart syntactically correct, but does not favor an automatic derivation toward formals methods. The automation of the derivation of figure 1 to the TPN (Time Petri Network) diagram uses two major functions, the translation function of a node toward TPN and the linking function of the adjacent nodes (Figure 7, lines 5, 14 and 16). The new node created must include all inputs and outputs associated with the nodes adjacent to it. The problem that arises is that, in this approach, the variation of the number of inputs or outputs gives new design elements. The question that emerges is: How to produce a finite set of transition rules of the semi-formal model to formal model from a set of infinite elements? In addition, the use of internal states with multiple inputs and outputs leads to under-utilization of the UML statechart nodes.

We propose in this work a rigorous approach to produce a UML statechart using patterns composition. We formalized the result by automatic translation toward Timed Colored Petri Nets. We also showed in our approach that it's possible to combined an UML statechart and an UML activity diagram to describe the dynamics of a system. Our work is organized in six sections. After this introduction, we present in Section 2 an overview of works related to the formalization of UML by derivation toward a formals methods. In Section 3 we present the principle of our machine for hot drinks, used as case study. Section 4 develops our patterns by showing through an example how they are composed to describe a system. In Section 5, we formalize our patterns by offering translation rules in Timed Colored Petri Net. We end our work by a conclusion followed by some prospects.

II. State of art

Software Engineering consists in proposing practical solutions, founded on scientific knowledge, in order to produce and maintain software with constraints on costs, quality and deadlines. In this field, it is admitted that the complexity of a software increases exponentially with its size [31][32]. To face these problems, todays mainstream approaches are build on the concept of Model Driven Engineering. Our work is based on formal analysis techniques based on models evaluations, specifically operational models, defined in terms of states and transitions such as TPN [24].The idea to couple the semi-formal and formal methods was introduced in the 90s under the name of ”mixed approach” [14][5][4][33]. Our study is based on asynchronous models, primarily on approaches using TPN. Most formal methods for the expression of the dynamic behavior are based on descriptions that use ”state-transition” system. However, the TPN are among the statechart formalism with powerful mechanisms of abstraction and description. A complete state of art of all work done in the field would be illusory. UML and Petri Nets are two specification techniques recognized in software engineering. Their coupling is motivated by the wish to use them together in a software development process that integrates both structural aspect and precision. The research results concerning the derivation of UML towards formal methods began with the thesis of Nguyen [21]. The main reason for this derivation was to preserve the achievements of the semi-formal methods already widespread, and strengthen with a formal point of view, without requiring a reconstruction of the system. Several teams have worked on the coupling between UML and formal languages [11][10][13].

Lilius et al [17] propose a formalization of UML statechart diagram for verification by the model-cheking method. The authors use PROMELA which is an algebra process with an asynchronous modeling paradigm, centered on the description of inter-process communications.

Attiogbé et al. [7] propose a generic approach to integrate formal data in UML diagrams. The main reasons are firstly to model the dynamic aspects of complex systems with a user-friendly language and graphics such as statechart of UML and secondly, to be able to specify formally and at a high level of abstraction, the data involved in these systems, using algebraic specifications.

At the University of Versailles Kamenoff [16] proposed an approach of derivation of UML toward B, by adding to the model the translation rules of the annotations OCL (Object Constraint Language). UML diagrams involved are class diagram and statechart diagram, enriched with the OCL constraints. The above works are all about the derivation of UML diagrams, that use as formal language either the algebraic process or the B method.

At the University of Chicago, Saldhana et al [27] [28], have worked on the derivation of class diagrams and state/transitions toward Colored Petri Nets. This approach uses an intermediate pattern ”the Object Petri Net Models (OPM)” as a gateway, allowing to produce the Petri Net model.

In [25][26], the authors have proposed a derivation of collaboration diagrams and states/transitions UML, toward the t-Timed Colored Petri Net, for the design of Real-Time systems. Their approach is based on the COMET method for the design of concurrent systems.

Bouchoul and Mostefai [9] proposed a formal approach for the specification of reactive distributed systems by rewriting logic. Their techniques consist of a twinning high level algebraic Petri Nets and a formal object-oriented language.

At the University of Paris 13, André et al [2] proposed an approach for the derivation of UML activity diagrams toward Colored Petri Nets, using composition of patterns. However, the approach has limitations to describe Real-Time systems. This limit is corrected in [3], where the authors proposed the derivation of the UML activity diagrams toward time Petri Nets for Real-Time systems, using composition patterns. Unlike the approaches discussed above, the proposal in [3] uses basic patterns with a single input and a single output, which by composition can produce composite objects that can have multiple inputs and outputs. In this paper, the approach is similar, but’s applied to elements of the statechart diagram and supports automatic derivation of this diagram toward the TPN. We have also show how we coupled the two UML diagrams (Activity and statechart) to assure an inter-diagram consistency.

III. Case study: The machine for hot drinks

We applied our proposals to the specification of a simple machine for hot drinks. The machine is made up of buttons which allows to serve tea or coffee, LEDs which indicates at every moment the state of the system (lack of water, in service) and sensor coupled to the tank which gives the volume of water available at every moment. At power on, the machine waits for the selection of a product. When a tea button is pressed, the system checks the water availability, the machine goes into service mode and the tea is served after 60 seconds. If it’s a coffee button, the system checks the availability of water, the machine goes into service mode and the coffee is served after 20 seconds. In case of water failure, the machine goes to the water defect state. If the machine makes 300 seconds without any button pressed, the system switches to standby mode. We have described in [3] this system, by a Composite Time Activities Diagram (CTAD). Figure 2 shows an improved version of this diagram. The inclusion of temporal aspects in the modeling of internal tasks in the states justifies our choice for timed-Colored Petri Nets. The selection of the type of product by the user generates other parameters, that are specific to this product such as the volume of water and the time required for it to be manufactured. These parameters are stored in tank of data and represent in our TPN modeling, the coloration of our tokens.

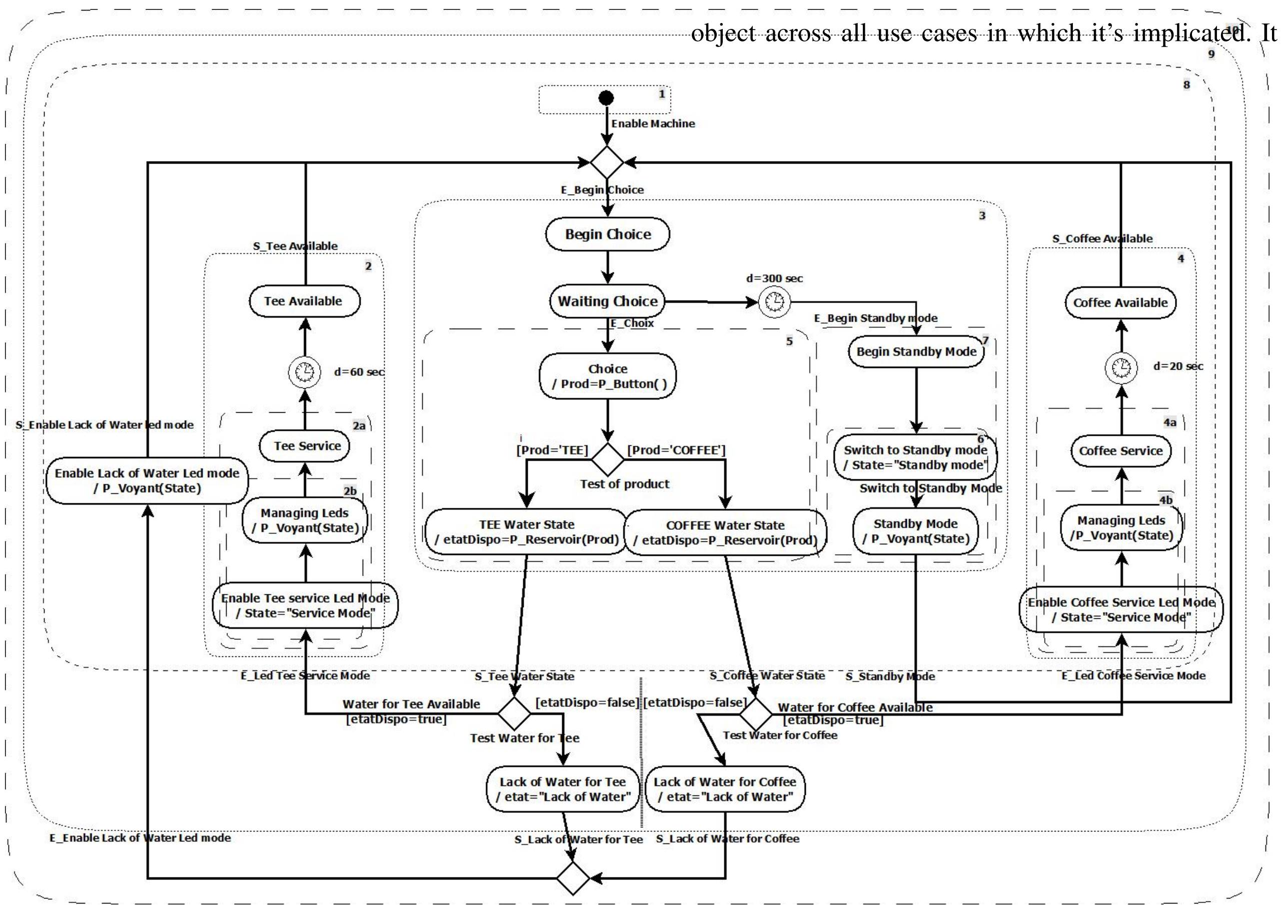


Fig. 2. CTAD of the machine for hot drinks

The overall behavior of the machine can be described by a combination of the UML statechart diagram and the UML activity diagram. Indeed, the internal activity of CSMD is described by a CTAD, present in figure 2.

IV. Our patterns statechart

UML (Unified Modeling Language) [23] has become the standard notation for object models in recent years. Its syntax and semantics are given in the natural languages, completed by assertions in OCL [22]. Although the UML meta-model provides a semantics, it's imprecise and insufficient to check the consistency of diagrams constituting the model. Such inconsistency problems can occur either at the intra-diagram level or at the inter-diagram level. In order to study such properties in UML diagrams dynamic, one of the methods is to provide a formal semantics for a part of UML. UML has several diagrams organized in multiple view and we will focus in this work on the dynamic view specifically to the UML statechart.

A. *UML statechart*

The UML statechart describes all possible states of an object. As most object-oriented methods, UML is based on Statecharts of David Harel [15]. Unlike the sequence diagrams that include all objects involved in a single use case, statechart diagrams indicate all state changes of a single object across all use cases in which it's implicated. It is a synthetic view of the dynamic operation of an object. However, the lack of rigor in its construction leads to models that do not facilitate the automation of the process of derivation toward a formal method (see paragraph 1). In [18], the authors proposed a redefinition of the semantics of the UML statechart elements. We present below the elements of the UML statechart as patterns of construction.

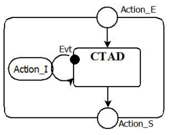
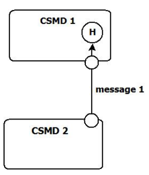
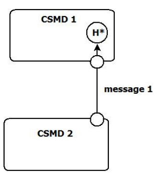
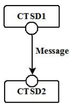
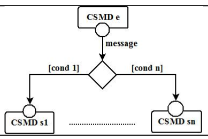
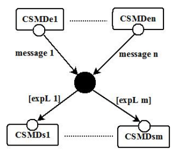
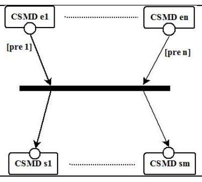
B. *Knowledge reused methods*

The increasing complexity of information systems and their more rapid evolution have motivated interest in the reuse models and methods [31] [32]. In this research, we have proposed modular and reusable patterns, encapsulated in objects that communicate through ports. We named them CSMD (Composite State Machine Diagram). Reuse of knowledge can be accomplished by several mechanisms [30]:

- Specialization: this is to adapt the knowledge provided by specialization for the system being modeled.
- The parameterizations: here, the entity to be reused is associated with a set of settings.
- The composition: here, the knowledge's provided are associated to each other coherently, to form a fragment of the complete system.

We proposed in this work a reused method which use the composition of designed patterns. Software pattern describes a problem frequently encountered in a particular context of development with a general proven solution for this problem [1] [12]. Several studies have focused on modeling by the

TABLE I
SEMANTIC OF THE PATTERNS OF THE STATECHART DIAGRAM ELEMENTS

Name of the node	Pattern notation	Semantic
Initial state	●	Marks the beginning of a process describing the internal elements of a system object. It is only used once in a diagram.
Final state	◉	Marks the end of a process describing the inner workings of an object of the system. It can be used many times in a diagram.
Internal state		Represents a node of a statechart diagram in which several actions and activities can be executed when it is active. At the entrance of the node, the action ”Action_E” is executed. At the end of it’s execution, the activity described by CTAD is executed. If during this execution, event represented by the black spot occurs, the action ”Action_I” corresponding is run. During this execution, the activity is momentarily stop. At the exit of the node, the action ”Action_S” is executed. If the message carried by an output transition of the node is valid, the internal activity is interrupted.
Shallow History		Activation of the message ”Message 1” disables the CSMD2 and enable the last state visited by the CSMD directly internal to CSMD1. Indeed, a transition that target the Shallow History is equivalent to a transition that targets the last state visited in the region containing the H.
Deep History		Activation of the message ”Message 1” disables CSMD2, and the last active simple state in CSMD1 recently used become enabled. If the internal state in CSMD1 is a composite CSMD, the last internal state becomes active, and so on through the hierarchy.
Transition		If the message carried by the transition is valid, the node upstream of the transition is deactivates and the node downstream of the transition is activated.
Choice node	 CSMD e message [cond 1] [cond n] CSMD s1 CSMD sn	If the message carried by the transition is valid, execution continues to the CSMDi for which [condition i] $(1 \leq i \leq n)$ is true.
Junction node	 CSMDe1 CSMDen message 1 message n [expL 1] [expL m] CSMDs1 CSMDsm	The validity of the messages ”messages i” to the outputs of the nodes CSMDei for $1 \leq i \leq n$ disables these active nodes and nodes CSMDsj for which the logical expression [Expl j] $(1 \leq j \leq n)$ is true.
Fork node	 CSMD e1 CSMD en [pre 1] [pre n] CSMD s1 CSMD sm	The validity of all messages ”[pre i]” $(1 \leq i \leq n)$ disables the CSMDei nodes and enable parallely the CSMDsj nodes for $1 \leq j \leq m$.

patterns. Applications are found for the modeling of business process [2] [20] and for the modeling of software processes [30], [3]. Here, we introduce the concept of CSMD for ”Composite State Machine Diagram”, which is the result of a complex UML statechart or composite diagram. It is built by the composition of the elements of our patterns. Figure 3 is a diagram of the CSMD. P_i $(1 \leq i \leq N)$ is the i^{th} entries of the CSMD and Q_j $(1 \leq j \leq M)$ is the j^{th} outputs, where N is the number of inputs and M is the number of outputs.

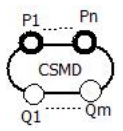


Fig. 3. Diagram of a CSDM

They are each associated with an elementary action.

The table 1 describes the elements of our proposal. It gives for each pattern the name of the diagram node,

the pattern notation and its semantics. The transition from one state to another is controlled by the validity of the carried message. A message will be valid if the event occurs and the related condition is true.

We have presented different elements of our proposal above. The transition from one node to another is controlled by the message that carries this transition.

C. Composition rules of our patterns

We define below the composition rules of our design patterns, so that we can used to build a composite CSMD.

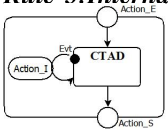
Rule 1: Initial state

● It’s a CSMD which does not have input and having itself as output.

Rule 2: Finale state

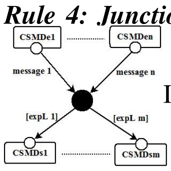
◉ It’s a CSMD which does not have output and having itself as input.

Rule 3:Internal state



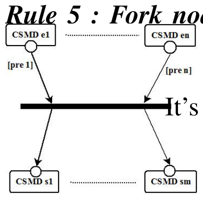
is an elementary CSMD having a single input and a single output.

Rule 4: Junction node

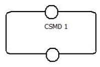
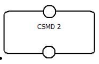
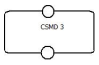


It’s a CSMD having as input the free inputs inherited from $CSMDe_i$ $(1 \leq i \leq N)$ and as output the free outputs inherited from $CSMDs_j$ for $(\leq j \leq M)$.

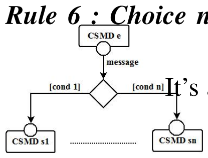
Rule 5 : Fork node



It's a CSMD having as input the free inputs inherited from $CSMDe_i$ $(1 \leq i \leq N)$ and as outputs the free outputs inherited from $CSMDs_j$ $(1 \leq j \leq M)$.

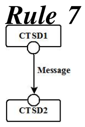
Consider  it's a CSMD having a clearly identified output.  and  are CSMD having clearly identified inputs

Rule 6 : Choice node



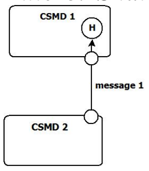
It's a CSMD having as input the free inputs inherited from CSMDe and as outputs the free outputs inherited from $CSMDs_i$ $(1 \leq i \leq n)$.

Rule 7 : Transition



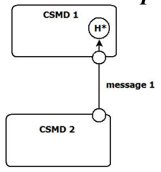
It's a CSMD having as input free inputs inherited from CSMD1 and as output free outputs inherited from CSMD2.

Rule 8: Shallow History



It's a CSMD having as inputs the free inputs inherited from CSMD2 and as outputs the free outputs inherited from CSMD1.

Rule 9: Deep History



It's a CSDM having as inputs the free inputs inherited from CSMD2 and as outputs the free outputs inherited from CSMD1

We can build more complex CSMD by applying the rules above.

D. CSMD of the machine for hot drinks

Figure 4 is a CSMD that models the overall operation of the machine for hot drinks. It is obtained by applying our CSMD composition rules. Each CSMD is described by a set of parameter and offer to its environment a set of input and output ports. The CSMD5 models the operation of our machine for hot drinks. It's built using the CSMD4 and the initial state, by applying the composition rule 4. The CSMD4 models the operations of the sub-machine that supports the preparation of the product and the management of the lack of water. It is built from CSMD1, CSMD2 and CSMD3, by applying the composition rule 6.

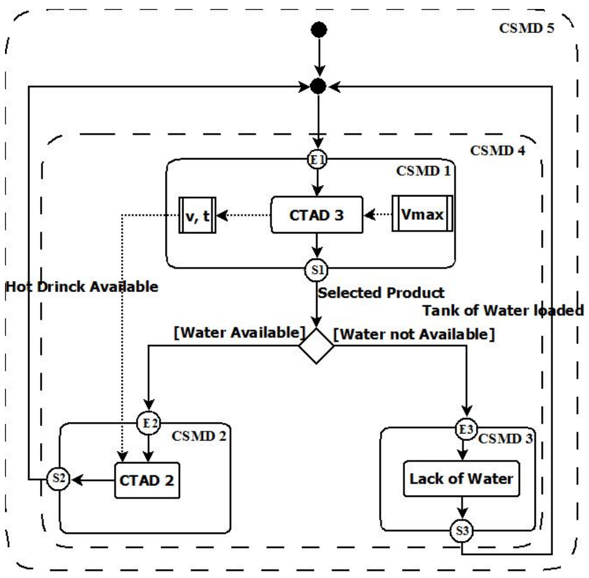


Fig. 4. CSMD of machine for hot drinks

The CSMD 1, 2 and 3 perform the activities modeled in Figure 2 by the CTAD2, CTAD3 and "Lack_of_Water". The entries in our composite states are named Ei and the outputs Sj, they materialize the actions performed when one crosses the port. We give below a list of actions associated with each port.

- E1: Light the LED: machine is enable.
- E2: Light the LED: machine is in service.
- E3: Light the LED: lack of water indicator.
- S1: Switch off the LED: machine ready.
- S2: Switch off the LED: machine in service.
- S3: Switch off the LED: lack of water.

The task performed by the CSMD1 consumes the data "Vmax" and produces the data, "t" and "v" consumed by the task performed by the CSMD2.

V. Formalization of the statechart diagram

This section focuses on the translation of our UML statechart patterns. After justifying the choice of Petri Network type used in our work, we propose the translation of our modeled elements in this formalism. A translation of our machine for hot drinks is provided at the end of the section.

A. Time Colored Petri Networks

The use of non-formal approaches which is de facto standards as UML of the OMG (Object Management Group) have many shortcomings, particularly in terms of formal semantics, which seriously undermines the implementation of a formal verification of UML diagrams. We have used in this work as mentioned in the introduction a translation approach that consists of deriving the statechart diagrams toward the Timed Colored Petri Network. Several reasons have guided our choice of Petri Nets. They are widely used for the verification and validation of the systems. The Time

Colored Petri Nets allow to quickly get executable models; they provide the opportunities of simulation, verification and evaluation of performance very early in the development cycle. The temporal aspect of our Petri Net is associated with the use of the composite activity diagrams proposed in [3] for the modeling of the actions performed in the state of our statechart diagram. We have proposed in this work the UML patterns activity diagrams, for the modeling of real-time systems. We introduced delay operations for a fixed time, variable time and the limited time to complete the semantics of classical UML transition (see Figure 2). Our case study focuses on a simple machine for hot drinks. The machine allows us to prepare a type of product selected by the user. Parameters used to select the product will be the coloring carried by the tokens in our Petri Net. We have given below a formal definition of the Petri Net used.

Definition: A Time Colored Petri Net [35] is a tuple (P, T, C, Pre(), Post(.) M_0; I) where

- P is a finite set of places,
- T is a finite set of transition, with $T \cap P = \phi$,
- $Pre(.) \in (N^P)^T$ is the upstream incidence function,
- $Post(.) \in (N^P)^T$ is the downstream incidence function,
- M_0 is the initial marking,
- C is the color function,
- $I : T \longrightarrow J$ is a function that associates to each transition a static interval of the operation duration of a token in a place.

B. Translation of the UML statechart patterns toward Time Colored Petri Nets

Table 4 shows the transformation rules of our case study in Colored Petri Net. By applying these rules, it is possible to transform each CSMD toward the Colored Petri Nets. The notation Tr(CSMDi) will be used in the sequel like the translation in the Colored Petri Nets of the CSMDi.

The initial state is translated by a place carrying a token connected to a transition.

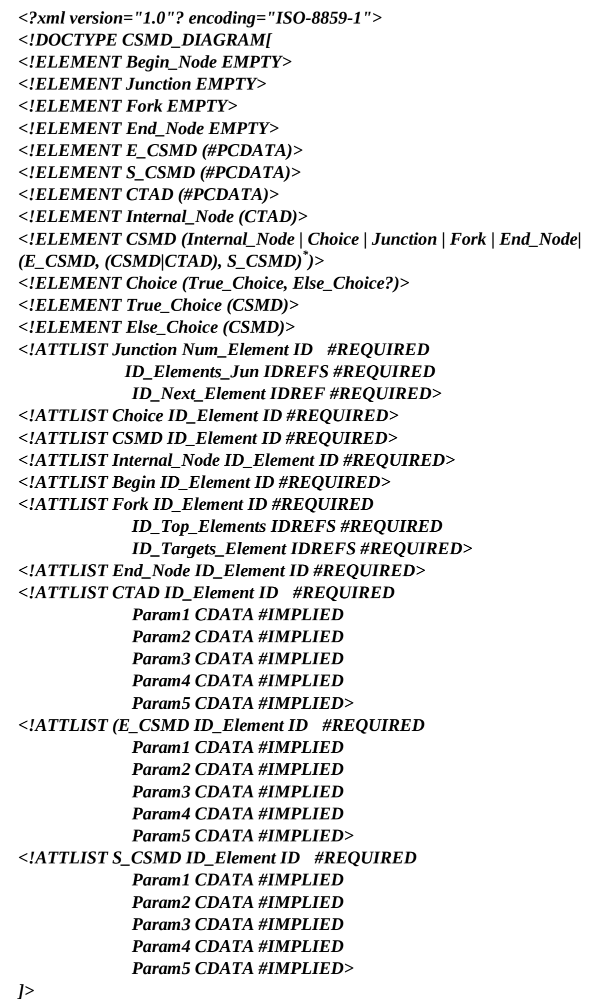
The end node is transformed into a place.

The internal state is transformed into Petri Net by using three places and three transitions. Two simple places, respectively model the input and output in the state and, one composite place which is the model of the activity performed in this state. The input task of the state is carried by the transition node of the input place, while the output task is carried by the transition node of the output place.

The transition loop allows to manage the temporary interruption of the activity.

The local history is managed by a place, from where all the internal states to the composite state in description converged. From each output of the state in description, the number of the last state is memorize in the variable ”h”, while the variable ”live” set to ”true” indicates the output of this state. When returning to the state in description, the variable ”live” changes to ”false” and execution continues in the ”h” state.

The deep history is managed by a place, from where all the internal states to the composite state in description, including its sub-states converged. In each output of the state in description, the number of the last active state including its active sub-states is stored in the variable ”h” while the variable ”live” is set to ”true” in order to indicate the output in the state. To return to the state in description, the variable ”live” is changes to ”false” and execution continues in the ”h” state.



```
<?xml version="1.0"? encoding="ISO-8859-1">
<!DOCTYPE CSMD_DIAGRAM[
<!ELEMENT Begin_Node EMPTY>
<!ELEMENT Junction EMPTY>
<!ELEMENT Fork EMPTY>
<!ELEMENT End_Node EMPTY>
<!ELEMENT E_CSMD (#PCDATA)>
<!ELEMENT S_CSMD (#PCDATA)>
<!ELEMENT CTAD (#PCDATA)>
<!ELEMENT Internal_Node (CTAD)>
<!ELEMENT CSMD (Internal_Node | Choice | Junction | Fork | End_Node|
(E_CSMD, (CSMD|CTAD), S_CSMD)*)>
<!ELEMENT Choice (True_Choice, Else_Choice?)>
<!ELEMENT True_Choice (CSMD)>
<!ELEMENT Else_Choice (CSMD)>
<!ATTLIST Junction Num_Element ID   #REQUIRED
               ID_Elements_Jun IDREFS #REQUIRED
                ID_Next_Element IDREF #REQUIRED>
<!ATTLIST Choice ID_Element ID #REQUIRED>
<!ATTLIST CSMD ID_Element ID #REQUIRED>
<!ATTLIST Internal_Node ID_Element ID #REQUIRED>
<!ATTLIST Begin ID_Element ID #REQUIRED>
<!ATTLIST Fork ID_Element ID #REQUIRED
                ID_Top_Elements IDREFS #REQUIRED
                ID_Targets_Element IDREFS #REQUIRED>
<!ATTLIST End_Node ID_Element ID #REQUIRED>
<!ATTLIST CTAD ID_Element ID   #REQUIRED
                Param1 CDATA #IMPLIED
                Param2 CDATA #IMPLIED
                Param3 CDATA #IMPLIED
                Param4 CDATA #IMPLIED
                Param5 CDATA #IMPLIED>
<!ATTLIST (E_CSMD ID_Element ID   #REQUIRED
                Param1 CDATA #IMPLIED
                Param2 CDATA #IMPLIED
                Param3 CDATA #IMPLIED
                Param4 CDATA #IMPLIED
                Param5 CDATA #IMPLIED>
<!ATTLIST S_CSMD ID_Element ID   #REQUIRED
                Param1 CDATA #IMPLIED
                Param2 CDATA #IMPLIED
                Param3 CDATA #IMPLIED
                Param4 CDATA #IMPLIED
                Param5 CDATA #IMPLIED>
]>
```

Fig. 5. DTD of our XML code

The transition node between Tr(CSDM) is translate by a waiting place and a transition node carrying the message that controls the crossing toward the Tr(CSMD) following.

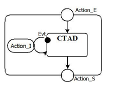
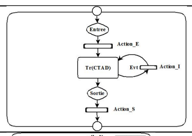
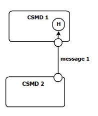
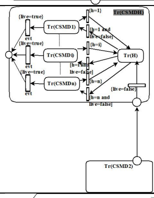
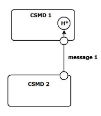
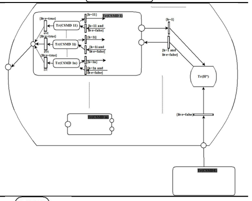
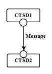
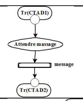
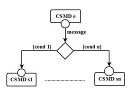
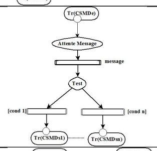
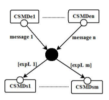
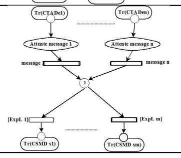
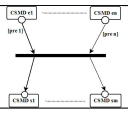
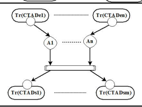
The junction node is represented by a place that is the destination of all arcs to merge, followed by a transition node.

The decision node is obtained by a place call ”test”, connected to a set of transitions, each carrying a guard. We use this translation to check the volume of water in our case study.

The synchronization node is transformed by a single transition between the composite places downstream and the composite places upstream.

Our model is stored as a XML (Extensible Markup Language) file (fig. 6)[39], that is validate by a DTD (Document Type Definition) (fig. 5).

TABLE II
TRANSLATION OF OUR PATTERNS INTO PETRI NETS

Type of the elements	UML representation	Translation into Petri Nets
Initial state	●	
Final state	◉	○
Internal state		
Shallow History		
Deep History		
Transition		
Choice node		
Junction node		
Fork node		

Many translation algorithm on Petri Nets have been proposed in the literature [40][41]. The algorithm (Fig. 7) implements our translation rules above. This algorithm uses as data structures, a pointer to the first node of the input diagram and a list to store the nodes through which we have already past. The functions ”Makenode( )” and ”Linknode( )” are respectively used to transform a node in PN and to establish the link between current node and its adjacent nodes.

C. Translation of the case study in Petri Nets

Figure 8 presents the translation in Time Colored Petri Net of our machine for hot drinks.

The composite place Tr(CSMD5) is obtained by joining the node Tr(CSMD4) and the initial state to the place ”J”. Tr(CSMD4) is constructed by applying the transformation rule of the decision nodes. The place ”Test1” interconnects Tr(CSMD1), Tr(CSMD2) and Tr(CSMD3). The CSMD1, CSMD2 and CSMD3 are states such as shown in Table 4. At the entrance, respectively at the exit of each states, the actions ”E1” and ”E2” respectively ”S1” and ”S2” are executed. The activities implemented in the state are described by

```
<?xml version="1.0"? encoding="ISO-8859-1" standalone="no">
<CSMD_Diagram>
    <Begin ID_Element="4"/>
    <Junction Num_Element="2" ID_Element_Jun="2 3 4"  ID_Next_Element="1"/>
    <CSMD Num_Element="1"  >
        <E_CSMD Num_Element="11"  Param1="ON" Param2="Machine Enable">
            Led_Control_Procedure
        </E_CSMS>
        <CTAD Num_Element="12" Param1="VMAX" param2="?" Param3="?">
            Generate_v_t_Data
        </CTAD>
        <S_CSMD Num_Element="13" Param1="OFF" Param2="Machine Enable">
            Led_Control_Procedure
        </S_CSMS>
    </CSMD>
    <Choice Num_Element="5" >
        <True_Choice>
            <CSMD Num_Element="2">
                <E_CSMD Num_Element="21"  param1="ON" Param2="In to
                Service">
                    Led_Control_Procedure
                </E_CSMS>
                <CTAD Num_Element="22"  Param1="t" Param2="v">
                    Prepare_Product
                </CTAD>
                <S_CSMD Num_Element="23" Param1="OFF" Param2="In to
                Service">
                    Led_Control_Procedure
                </S_CSMS>
            </CSMD>
        </True_Choice>
        <Else_Choice>
            <CSMD Num_Element="3">
                <E_CSMD Num_Element="31" param1="ON" Param2="Lack of
                Water">
                    Led_Control_Procedure
                </E_CSMS>
                <CTAD Num_Element="32" >
                </CTAD>
                <S_CSMD Num_Element="33" param1="OFF" Param2="Lack of
                Water">
                    Led_Control_Procedure
                </S_CSMS>
            </CSMD>
        </Else_Choice>
    </Choice>
</CSMD_Diagram>
```

Fig. 6. XML code of the case stordie model

```
void Transformer(int begin)  //"begin" represent the first node of the structure
{
        enfiler(begin);
        Marquer[begin]=1;
        Makenode(begin);    //transforming a node "begin" that represente the UML
                            // element to a RdP node by applying our rules
while (file not empty)
{
    s=defiler() ;
    for all w adjacent to s do
    {
        if(Marquer[w]==0)
        {
          Marquer[w]=1;
          enfiler(w);
          Makenode(w)  //tansforme the node "s" that represent the UML
                       // element to a RdP node by applying our rules
        }
        Makelink(s,w)    // establish the link between the current node and the nodes
                         // adjacent to it.
    }
  }
}
```

Fig. 7. Algorithm for the automatic transformation

the translations of the CTAD: Tr(CTAD2), Tr(CTAD3) and Tr(Lac_of_water). The task Tr(CTAD3) has a place named choice and a place named ”water_state”. The place ”choice” have the type ”product” and each of its tokens would be colored by (prod, v, t) representing the selected product, the volume of water and the time required for its manufacture. The Place ”water_state” has the type ”Param_Product” and each of its tokens would be colored by (prod, v, t, v max), which will be transmitted in the network. The coloration vmax gives the current volume of water in the tank. The Petri Net diagram (fig. 6) that models our machine for hot drinks is used as input to a model checker supporting a formal verification of temporal properties. This check can be done using tools such as TINA [8] or Romeo [19].

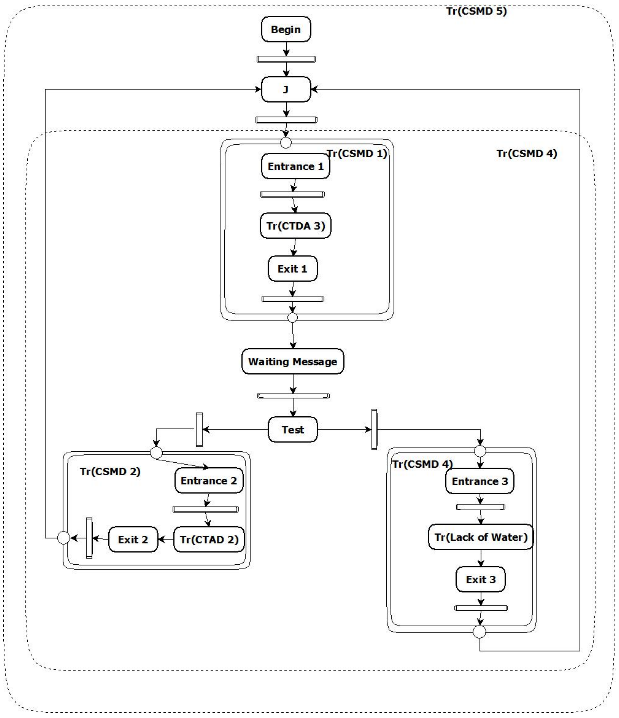


Fig. 8. Petri net diagram of the machine for hot drinks

VI. Conclusion

We have proposed in this work an automatic approach to formalize the UML statechart diagram by translating it toward the time Colored Petri Nets. Firstly, our method consisted to proposing statechart diagram elements patterns, associated with a set of composition rules to obtain a more rigorous design approach. By applying the composition rules of our statechart diagram patterns, it is possible to compose design objects more voluminous that we called CSMD. Secondly, we formalized our patterns by automatic translation of its elements in the time Colored Petri Nets. We applied our proposal for modeling a simple machine for hot drinks. Through this example, we show how our patterns composition rules and the transformation method into the Petri Nets is done. By using the Petri Nets model of our example as input of model checking tools such as TINA or Romeo the verification of some properties of the system is possible. The approach also shows a possible combination between statechart diagrams and activity diagrams. Indeed, the internal activity of CSMD is described by a CTAD. Our future works shall propose a new software development approach. using these patterns.

References

[1] Christopher Alexander, Sara Ishikawa, and Murray Silverstein. A pattern language : towns, buildings, construction. Oxford University Press, New York, 1977. Second volume d'une saarie de trois livres (Vol. 1 The timeless way of building, vol. 3 The Oregon experiment).

[2] Étienne André, Christine Choppy, and Gianna Reggio. Activity diagrams patterns for modeling business processes. Technical report, 2013.

[3] Étienne André, Christine Choppy, and Thierry Noulamo. Modelling timed concurrent systems using activity diagram patterns. KSE'14, Springer Advances in Intelligent Systems and Computing, octobre 2014.

[4] P. Andre, F. Barbier, and J. C. Royer. Une expérimentation de développement formel objet. Techniques et Sciences Informatiques, 24(8), June 1995.

[5] Pascal André. Méthodes formelles Objets pour le développement du logiciel : etudes et propositions. PhD thesis, Université de Rennes 1, 1995.

[6] Egidio Astesiano and Gianna Reggio. UML as heterogeneous multiview notation: Strategies for a formal foundation. In L. Andrade, A. Moreira, A. Deshpande, and S. Kent, editors, Proceedings of the OOPSLA'98 Workshop on Formalizing UML, ACM Press, 1998.

[7] C. Attiogbé, P. Poizat, and G. Salan. Intégration de données formelles dans les diagrammes d'états d'uml. Technical report, 2002.

[8] Bernard Berthomieu and Franois Vernadat. Time petri nets analysis with TINA. In Third International Conference on the Quantitative Evaluation of Systems (QEST 2006), 11-14 September 2006, Riverside, California, USA, pages 123-124, 2006.

[9] Faiza Bouchoul and Mohammed Mostefai. Vers une approche formelle pour la spécification des systèmes distribués réactifs. In L. Andrade, A. Moreira, A. Deshpande, and S. Kent, editors, Proceedings of the 4th International Conference on Computer Integrated Manufacturing CIP'2007, 2007.

[10] J.-L. Boulanger, G. Mariano, and P. Bon. Semi formal modelling and formal specification : Uml b in simple railway application. In ICSSEA 2003, CNAM, Paris, France, Novembre 2003.

[11] Jean-Michel Bruel and Robert B. France. Transforming uml models to formal specifications, 1998.

[12] J.O. Coplien. Software Patterns. SIGS, June 1996.

[13] Marc Frappier, Frédéric Gervais, Régine Laleau, Benot Fraikin, and Richard St Denis. Extending statecharts with process algebra operators. Innovations in Systems and Software Engineering, 4(3) :285-292, October 2008.

[14] Martin D. Fraser, Kuldeep Kumar, and Vijay K. Vaishnavi. Informal and formal requirements specification languages : Bridging the gap. IEEE Trans. Software Eng., 17(5) :454-466, 1991.

[15] David Harel. Statecharts : A Visual Formalism for Complex Systems. Science of Computer Programming, 8(3) :231-274, June 1987.

[16] Rafael M. Kamenoff. Spécification formelle objets en UML/OCL et B : Une approche transformationnelle. PhD thesis, Université de Versailles - PRiSM, December 2002.

[17] Johan Lilius and Iv'an Paltor. Formalising uml state machines for model checking, UML'99 - The Unified Modeling Language, doi: 10.1007/3-540-46852-8_31, page 756, url: http://dx.doi.org/10.1007/3-540-46852-8_31, 1999.

[18] Johan Lilius and Ivn Porres Paltor. The semantics of uml state machines. Technical report, 1999b.

[19] Didier Lime, Olivier H. Roux, Charlotte Seidner, and Louis-Marie Traonouez. Romeo: A parametric model-checker for petri nets with stopwatches. In Stefan Kowalewski and Anna Philippou, editors, 15th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2009), volume 5505 of Lecture Notes in Computer Science, pages 54-57, York, United Kingdom, March 2009. Springer.

[20] Ahmed Mekki, Mohamed Ghazel, and Armand Toguyeni. Validating timeconstrained systems using UML statecharts patterns and timed automata observers. In Proceedings of the Third international conference on Verification and Evaluation of Computer and Communication Systems, VECoS'09, pages 112-124. British Computer Society, 2009.

[21] H.P. Nguyen. Dérivation de Spécifications Formelles B Partir de de Spécifications Semi-Formelles. 1998.

[22] OMG. Response to the UML 2.0 OCL RfP, 2000. OMG doc. ad/2000-09-03.

[23] OMG. Unified Modeling Language Specification 2.0 : Superstructure, 2005. OMG doc. formal/05-07-04.

[24] C. A. Petri. Kommunikation mit Automaten. Technical report, 1962.

[25] RG Pettit and Hassan Gomaa. Modeling state-dependent objects using Colored Petri Nets. CPN 01 Workshop on Modeling of Objects, Components, and Agents, 2001.

[26] Robert G. Pettit IV and Hassan Gomaa. Validation of dynamic behavior in uml using colored petri nets. In PROC. OF UML2000 WORKSHOP - DYNAMIC BEHAVIOUR IN UML MODELS : SEMANTIC QUESTIONS, pages 295-302 Springer Verlag, 2000.

[27] J. Saldhana, S. M. Shatz, and Z. Hu. Formalization of object behavior and interactions from uml models. International Journal of Software Engineering and Knowledge Engineering (IJSEKE), 11(6), June 2001.

[28] John Anil Saldhana and Sol M. Shatz. Uml diagrams to object petri net models : An approach for modeling and analysis. In International Conference on Software Engineering and Knowledge Engineering, pages 103-110, 2000.

[29] The Precise UML Group. Untitled. http ://www.cs.york.ac.uk/puml/index.html, 1997.

[30] Hanh Nhi Tran. Modélisation de procédés logiciels base de patrons réutilisables. Thèse de doctorat, Université de Toulouse-le-Mirail, Toulouse, France, novembre 2007.

[31] Alina Zamula, Sergii Kavun, Complex systems modeling with intelligent control elements, International Journal of Modeling, Simulation, and Scientific Computing,doi: 10.1142/S179396231750009X

[32] Xiaodong Huang, Li Zhang, Jing Zhou, and Yaofei Ma, Multi-view collaborative modeling method for complex system, International Journal of Modeling, Simulation, and Scientific Computing, doi: 10.1142/S1793962316500306, Vol. 07, No. 03, September 2016.

[33] Mariusz Pelc, Analysis of Policy-Based Systems with AGILE Policies Using Petri Nets, International Journal of Software Engineering and Knowledge Engineering, doi: 10.1142/S0218194016500443, Vol. 26, No. 08 : pp. 1255-1283 October 2016.

[34] P. M. Merlin. A Study of the Recoverability of Computing Systems. PhD Thesis, Irvine, 1974.

[35] B. Berthomieu, F. Peres, and F. Vernadat. Bridging the gap between timed automata and bounded time petri nets. In FORMATS06, Springer LNCS 4202, pages 8297, 2006.

[36] Ahmed Kheldoun, Kamel Barkaoui , Malika Ioualalen, Formal verification of complex business processes based on high-level Petri nets, Information Sciences, Elsevier, 385-386 (2016), 39-54, 0020-0255, http://dx.doi.org/10.1016/j.ins.2016.12.044

[37] Liu, C., Zhang, F., Petri net based modeling and correctness verification of collaborative emergency response processes, Cybernetics and Information Technologies, Volume 16, Issue 3, 2016, Pages 122-136,

[38] Pettit, R.G., Gomaa, H., Fant, J.S.: Modeling and prototyping of concurrent software architectural designs with colored Petri nets. In: International Workshop on Petri Nets and Software Engineering. pp. 6779 (2009)

[39] T. Bray, et al., Extensible Markup Language (XML) 1.0, World Wide Web Consortium (W3C) Recommendation, 1998, http://www.w3.org/TR/1998/REC-xml-19980210 (valid by February 20, 2006)

[40] S. G. Wang, C. Y. Wang, and M. C. Zhou, A transformation algorithm for optimal admissible generalized mutual exclusion constraints on Petri nets with uncontrollable transitions, in Proc. IEEE Int. Conf. on Robotics and Automation, Shanghai, China, 2011, pp. 37453750.

[41] Z. Y. Ma, Z. W. Li, A. Giua, A Constraint Transformation Technique for Petri Nets with Certain Uncontrollable Structures, WODES14: 12th Int. Work. on Discrete Event Systems (Cachan, France), May 14-16, 2014. DOI: 10.3182/20140514-3-FR-4046.00085.