MODELING INPUT DATA CLASSES OF THE BUILDING ENERGY SIMULATION PROGRAM ENERGY PLUS WITH SCENARIOS AND HIERARCHICAL COLORED PETRI-NETS

A. BACHKHAZNADJI, A. BELHAMRI

Département de Génie Climatique, Faculté des Sciences de la Technologie
Université des Frères Mentouri Constantine, Algérie

Reçu le 10 Février 2014 – Accepté le 20 Mai 2014

Résumé

Le principal apport dans ce travail, outre le cadre méthodologique lié au processus itératif proposé, réside dans la mise en œuvre d'une nouvelle plateforme qui permet la création de prototype d'interface usager spécifique au fichier IDD d'entrée des données du programme de simulation des performances énergétique des bâtiments EnergyPlus.

La notation UML et les réseaux de pétri de hauts niveaux sont utilisés pour la première fois dans le processus de développement d'interface usager pour le programme EnergyPlus.

Cette méthodologie permet la génération des aspects statique et dynamique d'un prototype d'interface graphique à partir des spécifications de comportement et des données.

***Mots clés** : EnergyPlus, IDD File Scenarios, UML, réseaux de Petri colorés.*

Abstract

This paper addresses the problem of modeling input data classes of the input data dictionary (IDD) file of the Building energy simulation program EnergyPlus. Our endeavor is to provide an easy way for elaborating a behavior specifications of an interactive user interface intended for input data of the IDD file, using hierarchical colored Petri nets.

The modeling approach used in this work comprises two levels of abstraction: the use case level corresponding to the use case diagram model as defined in the unified modeling language (UML) and the scenario level as refinement of the former one. The color aspect of Petri-nets is used at the scenario level to preserve the independence of several scenarios after their integration.

The benefit of the approach consists in the structuring of the scenario acquisition and the approach of merging scenarios using composite color sets.

***Key words**: EnergyPlus, IDD File, Scenarios, UML, Colored Petri-nets.*

ملخص

المساهمة الرئيسية في هذا العمل ، بالإضافة إلى الإطار المنهجي المتصلة بعملية تكرارية محددة لملف معلوماتية مستخدم بإنشاء نموذج واجهة جديدة تسمح هو تنفيذ منصة المقترحEnergyPlus.

أداء الطاقة لبرنامج محاكاة IDDإدخال البيانات عملية تطوير في الأولى للمرة المستويات بيتري عالية وشباكات UMLاستخدام يتم EnergyPlus.

للبرنامج م معلوماتية مستخدم واجهة إنطلاقا من واجهة المستخدم هذه المنهجية تتيح لابراز الجوانب الثابتة والحيوية من نموذج .السلوكية والبيانات المواصفات.

كلمات مفتاحيه *: انرجي بلس، ملف ادد، سيناريوهات، يوم ل، شبكة الملونة بتري*

The need for formal techniques for analyzing systems is widely recognized, a large range of existing formalisms being in use for specifying systems. In modeling interactive systems, visual formalisms are required to reduce the gap between users and the analysts. Object-Oriented (O-O) methods like UML [1] offer one of these formalisms (statecharts). So far they only address the dynamic behavior of individual objects. The behavior of the overall system cannot be described explicitly; it must be synthesized from the statecharts of the objects of the system [2].

The main contribution of this paper is to provide an approach for the formal specification of the dynamic behavior of an overall new interactive system describing input classes of the IDD (Input Data Dictionary) file of the building EnergyPlus simulation program. We propose a process for deriving system specifications combining the UML as object-oriented method and colored Petri-Nets as formal technique [3]. At the beginning of the process, a use case diagram model is elaborated according to the UML 2.0 notation [4]. Then we transform this diagram onto Petri-nets having use cases as transitions and user interactions as guard conditions of transitions. Each transition of this net (use case) is then refined by a colored Petri-net constructed from scenario associated with various use cases. The choice of colored Petri-nets as formalism was directed by the support of concurrency and the notion of colored tokens which are crucial in scenarios integration. The use of colored Petri-nets [5] may allow for verification and simulation of the resulting specification.

In our previous work [6], we have investigated a UML (version 1.4) class diagramming model, which describe merely the static structure of the IDD input data classes and a model driven application had been implemented based on that diagram.

In this article, we aim to model the input data classes, mentioned in the IDD structure protocol manual [7], not only using the UML class diagram model but by introducing the UML use cases and scenario specifications. Scenarios have been identified as an efficient means for understanding user requirements [8] and for analyzing user interfaces. A typical process for requirements engineering based on scenarios, has two main tasks. The first task consists of generating scenario specifications which describe the dynamic behavior of the system. The second task concerns scenarios validation with users by simulation and prototyping.

Thus, we need Petri-nets objects that support hierarchies as well as token colors to distinguish between scenarios.

In this paper, we introduce in the subsequent sections, first: the formalism used to describe Petri-nets as a discrete-event modeling language. Second: we provide an overview of the UML 2.0, with special focus on the class diagram, use case diagram and sequence diagram models. Third: we present the approach which describes the process for deriving the input data dictionary system specifications and finally we conclude this paper.

1. COLORED PETRI-NETS

Colored Petri-nets (CPNs) is a language for the modeling and validation of systems in which concurrency, communication, and synchronization play a major role. Colored Petri-nets is a discrete event modeling language combining Petri-nets with the functional programming language Standard ML [9]. Petri-nets provide also the foundation of the graphical notation and the basic primitives for modeling concurrency, communication, and synchronization. Standard ML provides the primitives for the definition of data types, describing data manipulation, and for creating compact and parameterisable models. Data types can be atomic (integer, string, real, Boolean, enumeration), and structured (products, records, unions, lists, subsets). A CP-net model of a system is an executable model representing the states (places) of the system and the events (transitions) that can cause system to change state.

A CP-net model is also a description of a modelled system and it can be used as a specification or as a presentation. The behavioral aspect of a CP-net can be validated by means of a simulation and animation, and can be verified by means of more formal analysis methods, i.e. state spaces and place invariants [2]. The process of creating the description and performing the analysis, usually gives the modeler a dramatically improved understanding of the modelled system.

A simple CP-net model is defined as a bipartite graph consisting of places, transitions and tokens: CP-net model = < P, T, A, M>, where P: the set of places, T: the set of transitions, A: the set of directed arcs connecting places and transitions, M: the set of tokens (Marking of CP-net model) resident in places at a given moment. Each token has a data value attached to it. This data value is called the token color. It is a number of tokens and the token colors on the individual places which together represent the state of the system. In addition a CP-net may have an associated set of enabling and firing rules to establish under what conditions (particular marking) a transition is enabled.

Colored Petri-nets (CPNs) is defined as a group of:

$< \Sigma, D, P, T, A, \tau, G, E, I >$ where:

Σ: is a finite set of non-empty types called color sets,
D: is finite set of data fields,
P: is a finite set of places with $P \subseteq D$,
T: is a finite set of transitions with $D \cap T = \varnothing$,
A: is a set of finite arcs such that $A \subseteq PxT \cup TxP$,
τ: is a color function,

$$\tau: D \rightarrow \Sigma, \forall p \in P, \tau(p) = C \text{ and } C \in \Sigma,$$

G: is a guard function, $G: T \rightarrow expr$,
E: is an arc expression function E:

$$DxT \cup TxD \rightarrow expr,$$

I: is an initialization function, $I: D \rightarrow expr$, where I (d) is a closed expression and

$$\forall d \in D: [type (I (p)) = \tau(p)].$$

In the above definition [9], arc expressions specify tokens which are being added or removed by transitions. Both places and variables of expressions are typed and the initial marking is defined by the initialization function I.

2. UNIFIED MODELING LANGUAGE (UML)

The UML represents the unification of the best known object-oriented methodologies to provide a standard in the field of object oriented analysis and design. The UML does not provide a process for developing software, but it gives a syntactic notation to describe all parts of a system (data, function and behavior) through a number of diagrams:

- Functional or interactive view, described with the assistance of use case diagrams, sequence diagrams, and collaboration diagrams.
- Structural or static view, represented with the assistance of class diagrams, object diagrams, component diagrams, and deployment diagrams.
- Dynamic view, expressed by statechart diagrams, and activity diagrams.

In what follows, we first discuss UML diagrams that are relevant for our approach: use case diagram, sequence diagram, and class diagram.

2.1. Class diagram : IDDClassD

The IDDClassD represents the static structure of the input data structure of the IDD file of the EnergyPlus program. It identifies all the classes for the proposed system and specifies for each class its attributes, operations and relationships to other objects. Relationships include inheritance, association and aggregation. The IDDClassD is the central diagram of UML modeling. Figure 1 depicts the IDDClassD of the proposed static model. To building the objects (classes) model, we adopted the goal oriented approach [10]. The principle of this approach is to identify classes from use cases' goals rather than use case descriptions with the use case-driven process [11]. Classes, in the IDDClassD, are the entities that participate in achieving the goals allowing a user to interact with the input classes described in the IDD file. They have their own features and can collaborate with use cases.

The IDDClassD class diagram shows classes and relationships among them. In the example of figure 1, the kinds of relationships are associations. Four classes identify the structure of the diagram: a group class (*ObjectsGroup*), an input Object class (*InputObject*), a Numeric (*NumField*) and alphanumeric (*AlphaField*) classes. The two remaining ones, in the diagram, identify an actor/user (module developer) and an interface (ICS) classes. The association *Associate* shows the relationships between instances of the two classes: *ObjectsGroup* and *InputObject*. A multiplicity (1 to 1...*) is added to both ends of the association to point at the role of each class to the other. In the class diagram, an instance of the *ObjectsGroup* class may possess at least one instance of the *InputObject* class.

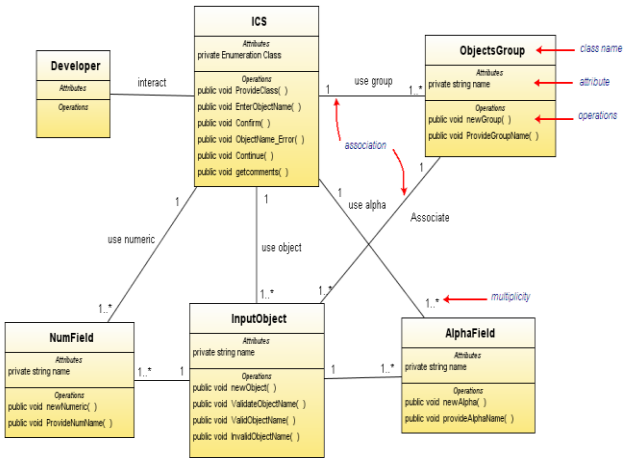

Figure 1 : The proposed IDDClassD of the IDD file

The use case diagram is a contribution of Ivar Jacobson in UML. A use case diagram describes the interactions between external actors and the system being modelled. It describes the sequence of actions carried out by a system with an aim of offering a service to the actors. A use case is a summary of scenarios for a simple task or goal. An actor (user) is who or what initiates the events involved in that task. Use cases are represented as ellipses, and actors are depicted as icons connected with solid lines to the use cases with which they interact. One use case can call upon the services of another use case. Such a relation is called an include relation, and its direction does not imply any order of execution.

Figure 2 shows the proposed use case diagram IDDUseCaseD. There is an actor (i.e. module developer) interacting with five basic use cases: *Identify_object*, *Generate_object*, *Edit_object*, *Delete_object*, and *Save_object* to IDD file. The use case *Generate_object*, call upon the service of the use case *Identify_object*, and include it as a subtask. An identification of an input object class may be a group of related input objects, or an input object, or a data field.

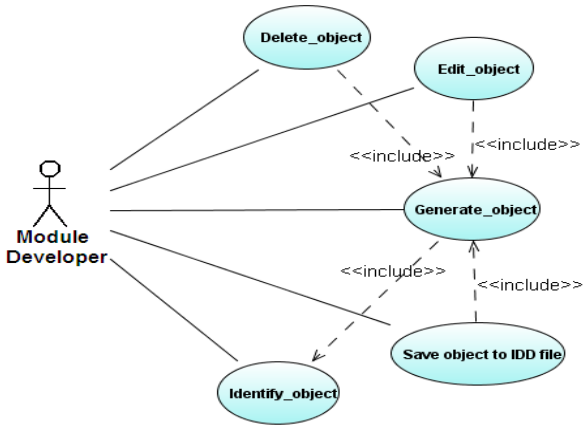

Figure 2 : The proposed IDDUseCaseD of the IDD file.

Furthermore, the UML comprises the extend relation, which can be considered as a variation of the include relation, as well as generalization relation which indicates that a use case is a special kind of another use case [1]. Use case diagram is helpful in visualizing the context of our application domain and the boundaries of the whole of the interface behaviour. An execution of a use case will typically be characterized by multiple scenarios (section III.2).

2.2. Sequence diagram ; IDDsequenceD

A scenario is an instance of a use case. It describes a group of interactions between actors and objects of the systems. In UML, scenario can be represented in the form of collaboration diagrams and/or sequence diagrams. Both types of diagrams rely on the same underlying semantics, and conversion of one to the other is possible [12].

For our work, we have chosen to use sequence diagram for their simplicity. A sequence diagram shows the interactions among the objects participating in a scenario in temporal order. It depicts the objects by their lifelines and shows the messages they exchange in time sequence. However, it does not capture the associations between objects. A sequence diagram has two dimensions: the vertical dimension represents time, and the horizontal dimension represents the objects. Messages are shown as horizontal solid arrows from the life line of the object sender to the life line of the object receiver.

Figure 3 depicts two sequence diagrams (two scenarios) of the use case Generate_object. Figure 3(a) represents a scenario where the developer is correctly entering a valid input object class name (*regularGeneration*), whereas figure 3(b) shows the case where an invalid input object class name is entered (*errorGeneration*).

The two scenarios are based on the input/output reference guide of EnergyPlus (EnergyPlus, v2.0 or latter). Thus, defining an input data entry in the IDD file [7], the following rules apply:

- A class name must be unique;
- The maximum length for an object name is 100 characters;
- Not an empty name.

3. DESCRIPTION OF THE MODELING APPROACH

In this section, we describe the process for obtaining the formal specification of the dynamic behavior of the IDD system to be modeled. We should emphasis that we address the behavior of the entire system and not just the behavior of its constituent objects.

For this purpose, we have used the two last kinds of UML diagrams (IDDUseCaseD and IDDSequenceD diagrams), described in previous section and combine them with colored Petri-Nets.

The approach is three steps course of action:

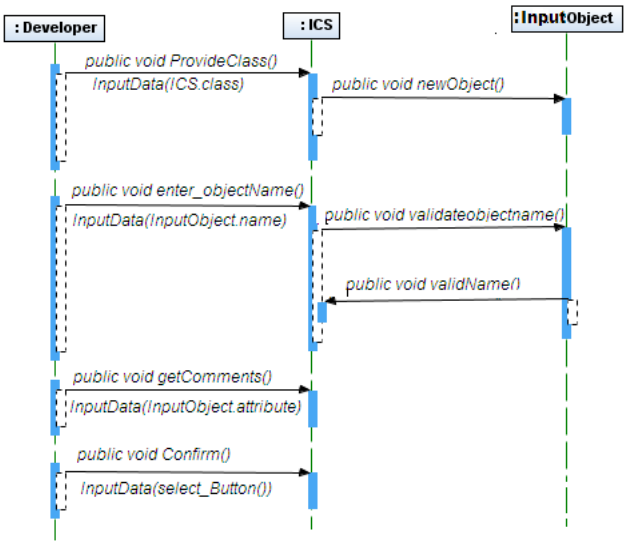

Figure 3(a) : Scenario *RegularGeneration* of the use case Generate_object.

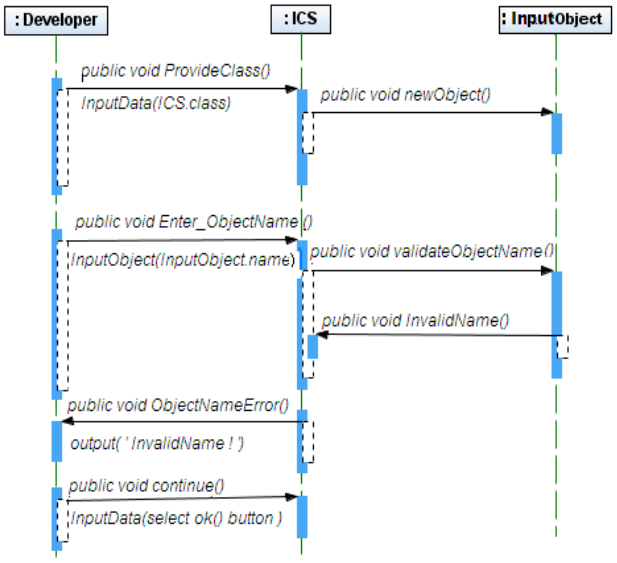

Figure 3(b) : Scenario *ErrorGeneration* of the use case Generate_object.

- The elaboration of the IDDUseCaseD of the IDD system, and the generation of the corresponding colored Petri-nets.
- The elaboration of several scenarios for each use case.
- The integration of scenarios by use case.

3.1. Generation of specification

This activity consists of deriving colored Petri-nets from both the acquired use case diagram IDDUseCaseD (see figure 2) and all sequence diagrams (see figure 3). These derivations are explained below in the subsequent subsections: use case specification and scenario specification.

3.1.1. Use case specifications

The Petri-net corresponding to the use case diagram is derived by mapping use cases into places (see figure 4). A transition (Enter) leading to one place corresponds to the initiating action (event) of the use case. A place *Init* is added to model the initial state of the IDD system. After a use case execution, the IDD system will return, via an (Exit) transition, back to its initial state for further execution of the use case. The place *Init* may contain several tokens to model concurrent executions. Figure 4 represents the Petri-net derived from the use case diagram.

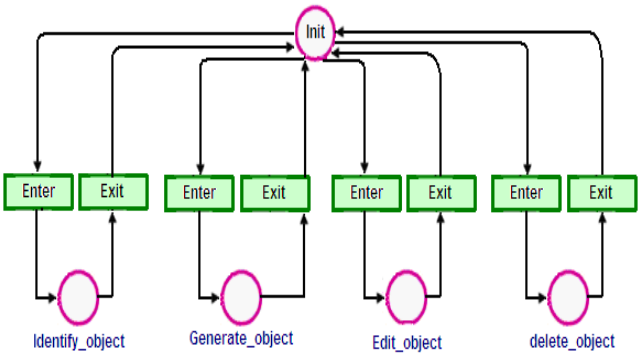

Figure 4 : Petri-net (PN) of the use case diagram specification

In a use case diagram, a use case can call upon the services of another use case with the relation *include*. The relation may have several meaning depending on the system being modelled. Consider two use cases: uc_1 and uc_2 (figure 5): the relation *include* between them may be interpreted in different ways [12]. Figure 5(a) gives the general form of this relation, UC_1 may be decomposed into three sub use cases: UC_{11} represents the part of UC_1 executed before the call of UC_2; UC_{12} is the part executed concurrently with UC_2 and UC_{13} is the part executed after termination of UC_2 (synchronization).

It is possible that two of these three use cases are empty, resulting in one of the configuration types shown in figure 5(b), figure 5(c), figure 5(d), figure 5(e), figure 5(f), and figure 5(g). The relation of type (g) between UC_1 and UC_2 means that UC_2 precedes UC_1; this implies that UC_1 is not directly accessible from the initial place (place *Init*). So, transition from the place *Init* to UC_1 must be changed to transition from UC_2 to UC_1. In the use case diagram IDDUsedCaseD (see figure 2), the relation *include* between the following couple of use cases is of type (g):

- (UC_1:Generate_object);
- (UC_2): Identify_object);
- (UC_1: Edit; UC_2: Generate_object);
- (UC_1: Save ; UC_2: Generate_object);
- (UC_1: Delete; UC_2: Generate_object).

The CPN Tools software which we adopted in our work, allows for the refinement of transitions, but does not support the refinement of places. Therefore, in order to substitute use cases, which are represented as places, for the colored PNs representing integrated scenarios (section IV.1.3).

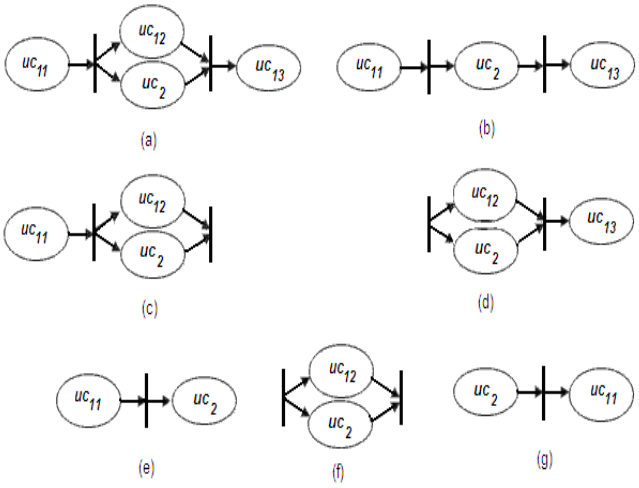

Figure 5 : The *include* relation and use cases.

The colored Petri-net obtained after processing the include relation requires adaptation: each subnet (Enter → $place_i$ → Exit) is substituted by a simple transition representing the use case, cf. dashed square in figure 6, and intermediate places, such as *enIdentify* and *endGenerate* are inserted (see figure 7).

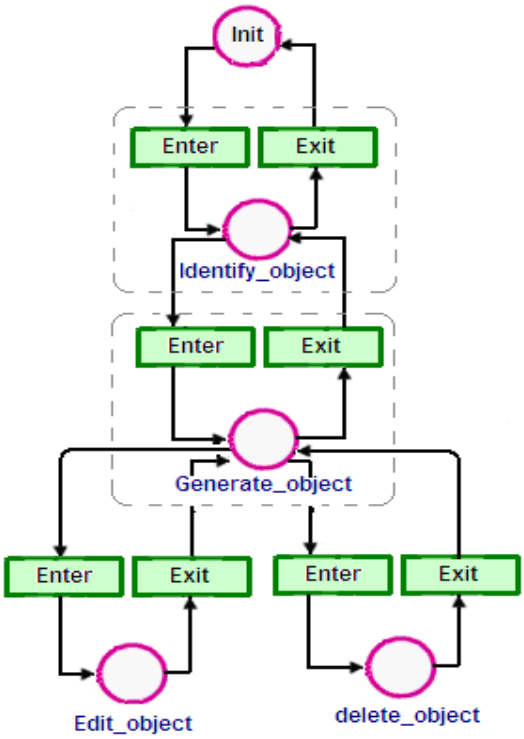

Figure 6 : PNs after processing the *include* relation.

The current marking of each place is indicated next to the place. The number of tokens on the place in the current marking is shown in the small circle, while the detailed token colors are indicated in the box positioned next to the small circle. Initially, the current marking is equal to the initial marking, denoted M_0. In the example of the figure 7, the initial marking has four tokens on place *Init* and one token on place *Sc_g*.

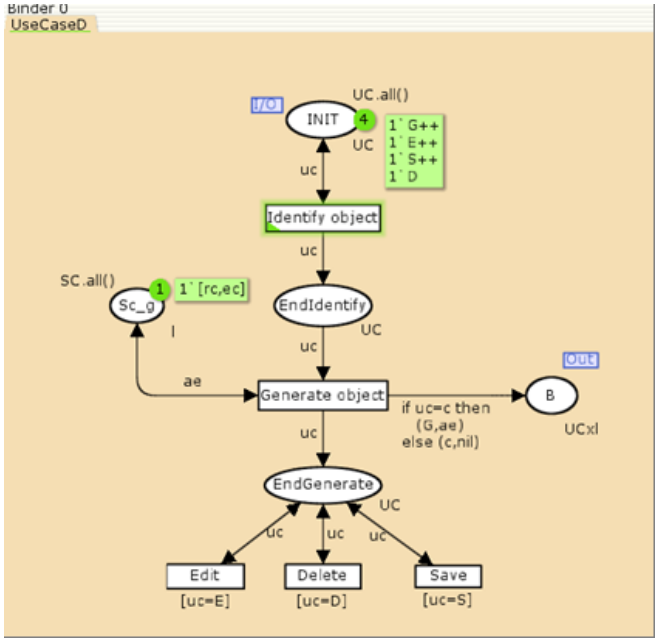

Figure 7 : Colored Petri-net of the IDDUseCaseD

Arc expressions which are the textual inscriptions positioned next to the individual arcs, and are built from typed variables, constants, operators, and functions. When all variables in an expression are bound to values (of the correct type) the expression can be evaluated. The arc expression evaluates to a multi-set of token colors (data values). Consider the arc expression *ae* (see figure 7) on the arc connected to both the transition *GenerateObject* and the place *Sc_g*. It contains the variable *ae* declared as:

var ae: l;
where l is declared as a color set of type list,
colset l = list SC;
and SC is a color set of type enumeration,
colset SC = with rc|ec;

Next to each place, there is an inscription which determines the set of token colors that places are allowed to have. The set of possible token colors is specified by means of a type and it is called the color set of the place. By convention, the color set is written below the place. In figure 7, places: *Init*, *EndIdentify*, and *EndGenerate* have the color set UC and it is defined in CPN ML programming language to be equal to the type enumeration:

Colset UC = with G|E|S|D;

This means that tokens residing on the three places will have an enumeration type as their token color. The color set UC is used to model the possible states of the colored Petri-net. A place and a transition may also be connected by double-headed arcs. A double-headed arc is shortened for two directed arcs in opposite directions between a place and a transition which both have the same arc expression. This implies that the place is both an input place and an output place for the transition. For instance, the transition *IdentifyObject* and the place *Init* are connected by double-headed arc (see figure 7).

3.1.2. Scenario specifications

For each scenario of the use case *Generate_object*, we construct an associated table of object state. This table is directly obtained from the sequence diagram of the scenario by following the exchange of messages from top to bottom, and identifying the changes in object state caused by the messages. In our example, tables 1 and 2 show the object states associated with the scenarios *regularGeneration* and *errorGeneration* (see figure 3).

In such tables, a scenario state is represented by a state vector of the objects participating in the scenario (column scenario state in tables I and II respectively).

From each object state table a colored Petri-net is generated by mapping scenario states into places and messages into transitions (see figures 8 and 9). Each scenario is assigned a distinct color, e.g., *rc* for the *regularGeneration* scenario, and *ec* for the *errorGeneration* scenario.

All colored Petri-nets (scenarios) of the same use case will have the same initial place (state) which we call B (see figures 8 and 9). This place will serve to link the integrated colored Petri-net (section IV.1.2) with the colored Petri-net modeling the IDD use case diagram depicted by figure 7.

Table 1 : Object state table associated with the scenario R*egularGeneration*

Object messages	Developer	ICS	Input Object	Scenario State
Provide Class	Present	Select Object	Void	S1
New Object	Present	Select Object	Object Generated	S2
Enter Object Name	Present	Prompt Name	Object Generated	S3
Validate Name	Present	Name Entered	Validate Name	S4
Valid Name	Present	Name Entered	Valid Name	S5
Get Comments	Present	Get Comments	Valid Name	S6
Confirm	Present	Confirm	Valid Name	S7

.
With :

S1= (Present, provide class, void).
S2= (Present, provide class, object generated).
S3= (Present, prompt name, object generated).
S4= (Present, name entered, validate name).
S5= (Present, name entered, valid name).
S6= (Present, get comments, valid name).
S7= (Present, confirm, valid name).

Table 2 : Object state table associated with the scenario *ErrorGeneration*

Object messages	Developer	ICS	Input Object	Scenario State
Provide Class	Present	Select Object	Void	S1
New Object	Present	Select Object	Object Generated	S2
Enter Object Name	Present	Prompt Name	Object Generated	S3
Validate Name	Present	Name Entered	Validate Name	S4
Invalid Name	Present	Name Entered	Invalid Name	S8
Error Name	Present	Error	Invalid Name	S9
Continue	Present	Continue	Invalid Name	S10

With:

S1= (Present, provide class, void).
S2= (Present, provide class, object generated).
S3= (Present, prompt name, object generated).
S4= (Present, name entered, validate name).
S8= (Present, name entered, invalid name).
S9= (Present, error, invalid name).
S10= (Present, continue, invalid name).

In the above tables a scenario state is represented by the union of the states of the objects participating in the scenario. In figures 8 and 9, the place 'B' represents the commencement of the use case. All scenarios related to the same use case have the same initial place but different token colors. This place will serve to link the integrated CPNs with the CPNs modeling the IDDUseCaseD of the IDD file (figure 4).

3.1.3. Scenarios integration

In this activity, we aim to combine all the CPNs corresponding to the scenarios of the use case UC_i, in order to produce integrated CPNs modeling the behavior of the use case. After integrating the two scenarios, the initial place B (see figure 10) will be shared, yet we do not know which scenario will be executed, and neither the token color *rc* nor the token color *ec* can be assigned to the initial place B. This problem was described by Elkoutbi and Keller and it was referred to it as interleaving problem [3].

To solve the interleaving problem, we introduce a composite color set, i.e., a token color that can take on several colors. Using CPN Tools [13]-[15]-[16] a color set is modeled by a list of colors. Upon visiting the places of the integrated CPNs, it will be marked by the intersection of its token colors of the place being visited. When the token passes to the place S4, it keeps the color set (*rc, ec)* to S5 its color changes to (*rc)* and will remain unchanged for the rest of its route, or it passes from the place S4 to the Place S8 its color changes to (*ec)* and will remain unchanged for the rest of its route.

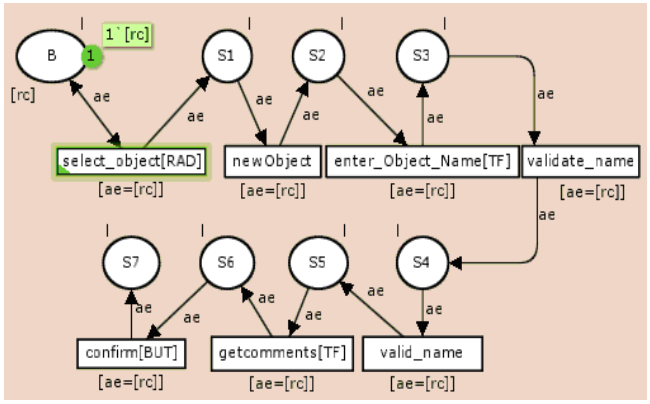

Figure 8 : A CPN corresponding to the *RegularGeneration* scenario.

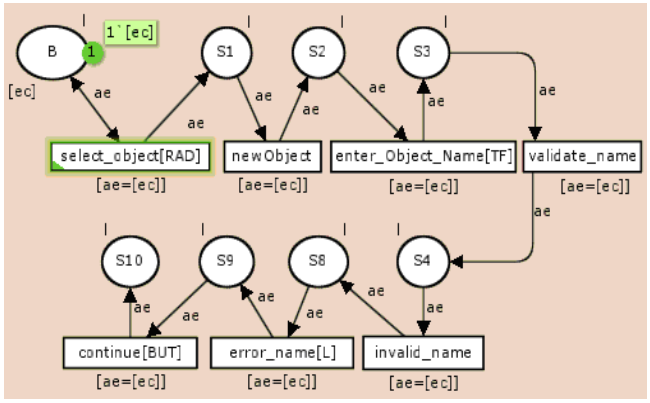

Figure 9 : A CPN corresponding to the *ErrorGeneration* scenario

Transitions that belong to only one of the scenario *RegularGeneration* or *ErrorGeneration* will be guarded by the token color of the respective scenarios (see transitions: (*getcomments, confirm*) and transitions: (*error_name, continue*). But this is not the case for transitions: (*valid_name and invalid_name*) which are required to transform colors from the color set (list of token colors) to a single color. Therefore they must be guarded by the composite color set. For transitions that are shared by the two scenarios *RegularGeneration* and *ErrorGeneration*, they will be guarded by the composite color set (see figure 10).

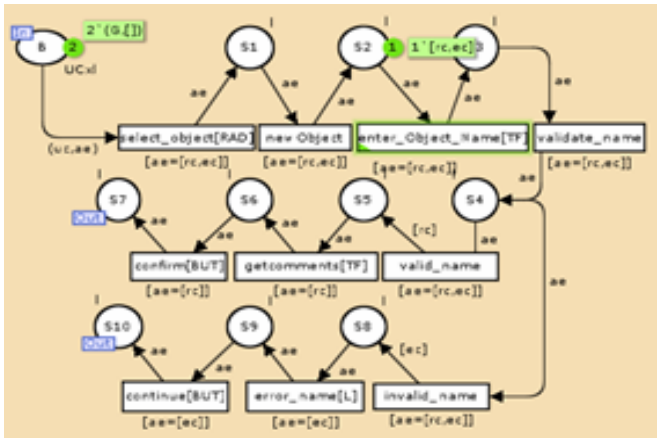

Figure 10 : Integrated CPN of the scenarios integration process.

Integrated colored Petri-net corresponding to a given use case can be connected to the colored Petri-net derived from the PN of the use case diagram (see figure 7) through a substitution transition *UseCase* (see figure 11), which is appended to the place B of the integrated colored Petri-net. This substitution transition will transform tokens of the colored Petri-net of the use case diagram to the composite color set of the integrated colored Petri-net. The colored Petri-net can be organized as a set of hierarchically related modules.

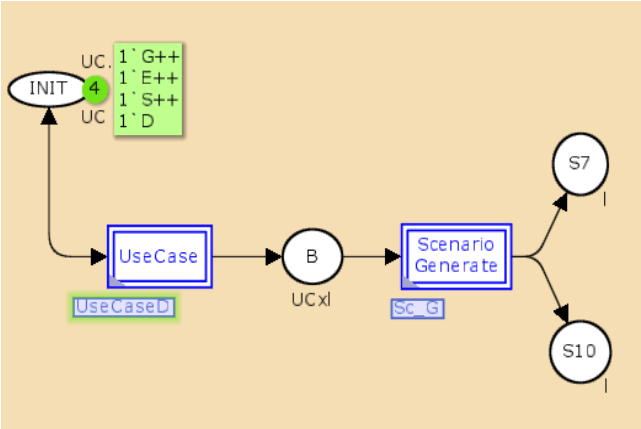

Figure 11 : A substitution transition UseCase graph.

4. INTERFACE PROTOTYPE GENERATION

In this activity, we derive from the colored Petri-net specification an interface prototype. The generated interface comprises a menu to switch between use cases which are directly accessible from the initial state (place Init) of the IDD interface. The various screens of the interface prototype represent the static aspect, the dynamic aspect of the interface Prototype, as captured in the colored Petri-net specification, maps into the dialog controls of the interface prototype. In our current implementation, prototypes are Java (an OOP language) application comprising each a number of frames and navigation functionalities, Fig 12.

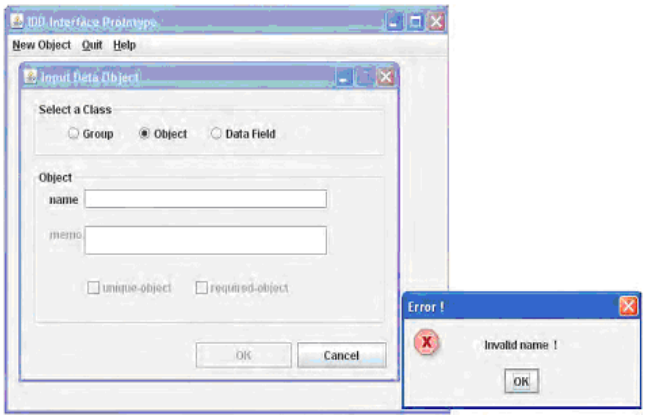

Figure 12 : Java implementation of the IDD interface prototype

For the purpose of IDD interface prototype generation, system view is appropriate when in all use cases and associated scenarios; only one actor interacts with the IDD interface. In the case of collaborative tasks (more than one user interacts with the use cases), however, an object view will be more suitable.

CONCLUSION

In this paper, we have proposed an approach for elaborating specifications of an interactive IDD system behavior by using two kinds of colored Petri-nets: a simple Colored Petri-Nets modeling the relation of the proposed IDDUseCaseD linked to several colored Petri-nets representing the use case behaviors as integrated CPNs which merge several scenarios to the same use case and preserves the independency between these scenarios after integration by means of color sets. There are many advantages with the proposed approach. First, the process of scenario acquisition is more structured.

Colored Petri-nets are among the most powerful visual formalism used for specifying complex and interactive system. Colored Petri-nets are known for their support of pure concurrency, all transitions having sufficient number of tokens in their input places may concurrently be fired. Colored Petri-nets in their basic form do not support hierarchy, but the extension of colored Petri-nets used in tools such as CPNTools software allows for hierarchies in the specification [16]. Non-deterministic choices can be more easily modelled using colored Petri-nets. When two or more transitions share the same input places, the system (CPNTools) chooses randomly to fire one of these transitions (section IV.1.3). Tokens that are specific to colored Petri-nets can be used both in controlling and simulating the IDD interface behavior and in modelling data and resources of that system. If the place Init of the Fig 7 contains only one token, the IDD interface system can just execute one use case at a time. When the place Init contains n tokens, n concurrent executions of different use cases are possible. It may be possible to execute n scenarios of the same use case (multiple instances).

In this work, we acquire scenarios use case by use case which we consider as more natural. Secondly, as a consequence of using Colored Petri-nets, the approach permits the modeling of concurrency between use cases, between scenarios and between copies of the same scenario. Finally, the approach solves the problem of interleaving scenarios by means of composite color sets.

As future work, we plan to pursue further development in particular the automation of the scenarios integration activity by introducing an algorithm which takes an incremental approach to integration. Given two scenarios with corresponding colored Petri-nets (CP-net1 and CP-net2); the algorithm will merge all places in CP-net1 and CP-net2 having the same names. The merged places will have as token colors the union of token colors of the two scenarios. Then, the algorithm looks for transitions having the same input and output places in the two colored Petri-nets and merges them to obtain the integrated colored Petri-net.

Références

[1] J. Rumbaugh, J. Jacobson, G. Booch, *The Unified Modeling Language, Reference Manual* (Addison Wesley Inc., 1999).

[2] I. Kriss, M. Ekoutbi, R.K. Keller, P.A. Muller, Automating the Synthesis of UML Statecharts Diagrams from Multiple Collaboration Diagrams, *In Bezivin ed., UML 98 Springer, LNCS 1618, Beyond the Notation*:132-147, *1999.*

[3] M. Elkoutbi, R.K. Keller, Modeling Interactive System with Hierarchical Colored Petri Nets, *In Proc. of Adv. Simulation Technologies Conf., pp. 432-437, Boston M.A., Soc. Comp. Simulation Intl HPC98 Special Session on Petri-Nets, 1998.*

[4] S. Sialhir, *Learning UML*, (O'Reilly Edition, 2003).

[5] L.M. Kristensen, Jorgensen, K. Jensen, Application of Colored Petri Nets in System Development, *Springer-Verlag, In Lectures on Concurrency and Petri Nets- Advances in Petri Nets, Proc. of 4th Advanced Course on Petri Nets, Vol. 3098 of Lecture Notes in Computer Science: 626-685, 2004.*

[6] A. Bachkhaznadji, A. Belhamri, A Model Driven Application for HVAC Energy Simulation Data: EnerMDA, *In Proc. of 3rd Mediterranean Congress of HVAC Engineering, Vol. 2, pp. 563-572, Climamed, Lyon France 20-21 November 2006.*

[7] EnergyPlus, *version 3.0,* a*pp1.eere.energy.gov/building/energyplus/*

[8] P. Hsia, J. Samuel, J. Gao, D. Kung, Y. Toyoshima, C. Chen, Formal Approach to Scenario Analysis, *IEEE Software, Vol. 11,(*Issue 2*): 33-41,1994.*

[9] K. Jensen, Colored Petri-Nets, Basic Concepts, Analysis Methods and Practical Use, *Practical use, Springer-Verlag Vol. 3, 1997.*

[10] L. Liang, From Use Cases to Classes, A Way of Building Object Model with UML, *Journal of Information and Software Technology, Vol.45*: *83-93,2003.*

[11] I. Graham, Use Cases Combined with Booch/OMT/UML, Process and Products, *Journal of Object Oriented Programming, 76-78, 1998.*

[12] M. Glinz, Statecharts for Requirements Specifications, As simple as Possible, as Rich as Needed. *Position Paper in the ICSE 2002 Workshop: Scenarios and state machine models, algorithms, and tools, Orlando, Florida, USA. 2002*.

[13] M. Elkoutbi, , I. Khriss, R. K. Keller, Automated Prototyping of User Interface Based on UML Scenarios. *Journal of Automated Engineering: 5-40, 2006.*

[14] M. Elkoutbi, R. K. Keller, User Interface Prototyping based on UML scenarios and High-level Petri-Nets. *Paper presented at the 21st International Conference on ATPN, Springer-Verlag LNCS 1825, pp.166-186, Aarhur, Denmark, 2000.*

[15] CPN Tools version 2.2.0. *Department of Computer Science, University of Aarhus 2006. Available at: http:// wiki.diami.au.dk/cpntools/.*

[16] *Jensen, K. Colored Petri-nets, Basic Concepts, Analysis Methods and Practical use. Volume3: Practical use, Springer-Verlag. 1997.*