7e Conférence Internationale de MOdélisation et SIMulation - MOSIM'08 - du 31 mars au 2 avril 2008 – Paris- France
« Modélisation, Optimisation et Simulation des Systèmes : Communication, Coopération et Coordination.».

SYNTHÈSE DE SCENARIOS EN DEVS

SQALI MAMOUN, LUCILE TORRES, CLAUDIA FRYDMAN

LSIS (Laboratoire des Sciences de l'Information et des Systems)
Université Paul Cezanne Aix-Marseille III
avenue Escadrille Normandie-Niemen13397 MARSEILLE CEDEX 20
{mamoun.skalli,lucile.torres,claudia.frydman}@lsis.org

RESUME : *Les scénarios et les machines à états sont deux modèles éprouvés, complémentaires, pour décrire le comportement d'un système réactif. Les scénarios représentent des exemples d'exécution du système sous la forme de séquences de messages échangés entre objets et constituent une vue partielle du système, alors que les machines à état permettent d'appréhender le comportement global du système. L'automatisation de la transformation des scénarios en machines à états apporte des éléments de réponse à différents problèmes comme la validation de la spécification du comportement, la vérification de la cohérence des scénarios, la génération automatique de tests à partir des scénarios. Dans cet article, nous proposons une méthode de traduction des scénarios en machines à états représentées dans le formalisme de spécification de système à événements discrets DEVS (Discret EVent System Specification). Les modèles DEVS induits représentent un comportement du système englobant tous les comportements exprimés par les scénarios et sont utilisés pour valider la spécification par simulation.*

MOTS-CLES : *spécification du comportement, synthèse, diagrammes de séquence, langages formels de scénarios, DEVS, validation par simulation.*

1. INTRODUCTION

L'ingénierie des exigences, activité qui intervient en amont du processus de développement, utilise largement les scénarios comme outil de définition des besoins et de construction des spécifications. Les scénarios représentent des exemples d'exécution du futur système, en termes d'échanges de messages entre le système et son environnement. Ils sont concrets et intuitifs pour un utilisateur et permettent de représenter partiellement les aspects comportementaux du système. De par leur caractère partiel, des scénarios peuvent être incohérents. L'utilisation d'un langage formel (avec une sémantique formelle) pour décrire les scénarios permet d'éviter les ambiguïtés et de détecter les incohérences. De plus, si le langage possède une représentation graphique, il est alors facilement compréhensible par les utilisateurs non familiers des méthodes formelles. Il existe différents langages formels de description de scénarios : les Message Sequence Charts (MSC) standardisés par l'ITU (ITU, 1996), parmi lesquels on distingue les MSC basiques (bMSC) et les MSC composés (hMSC pour High-Level MSC), les Live Sequence Charts (LSC) qui sont une extension des bMSC proposés par (Damm et Harel, 1999), les diagrammes de séquence d'UML 1.4 qui sont une version simplifiée des bMSC (Brian et Hans-Erik, 2004), etc.
L'utilisation des MSC est bien répandue en milieu industriel. Ce standard est utilisé aussi bien dans la conception de logiciels avec UML, que dans la conception de protocoles réseaux avec des techniques de description formelles telles que SDL (Reniers, 1995).
Les MSC permettent de décrire les exigences de manière incrémentale, notamment grâce aux hMSC conçus pour permettre la création de scénarios plus complexes en autorisant des compositions parallèles, des alternatives, des itérations, des séquences de bMSC, ainsi que la définition hiérarchique de scénarios. Comme pour les MSC, les hMSC bénéficient d'une syntaxe formelle normalisée, une exécution d'un hMSC étant obtenue à partir des exécutions des composants bMSC.
L'intérêt de l'utilisation des scénarios pour décrire le comportement du système est de réduire l'effort d'abstraction nécessaire à cette tâche de description, en ne concevant que des exemples d'exécution. Il est en effet facile d'imaginer les réactions du système à concevoir pour une séquence d'événements donnée. Il est au contraire difficile d'appréhender la complexité du comportement du système dans sa globalité. La difficulté réside dans la généralisation du comportement à toutes les séquences possibles. Cet article a pour objet d'induire à partir d'un ensemble de scénarios exprimés sous la forme de MSC, des modèles DEVS de spécification à événements discrets représentant le comportement global du système. La simulation de ces modèles doit produire les mêmes séquences d'événements pour les mêmes séquences en entrée que les scénarios et,

vraisemblablement, des séquences plausibles pour d'autres séquences en entrée que celles exprimées par les scénarios.

2. LES DIAGRAMMES DE SEQUENCE MSC

Dans cette section, nous décrivons brièvement les diagrammes de séquences de messages MSC. Nous présentons également un exemple simple employé pour illustrer notre approche. Cet exemple (voir figure 1) formé de plusieurs scénarios montre comment une unité de commande « Contrôleur » actionne une sonde qui détermine la profondeur de l'eau « Sonde » et un déclencheur automatique « Déclencheur » pour contrôler la pression d'une chaudière à vapeur. Une base de données « Bd » est employée comme unité de stockage pour protéger l'information de la sonde tandis que l'unité de commande « Contrôleur » exécute des calculs et envoie des commandes au déclencheur automatique.
Les MSC sont un formalisme qui permet de spécifier le comportement d'entités communicantes, auquel est associé une représentation graphique. Un diagramme MSC de base décrit le comportement de processus appelés « instances » qui communiquent de manière asynchrone. Les instances d'un bMSC sont représentées par un axe vertical, et les échanges de messages par des flèches de l'instance émettrice vers l'instance réceptrice, étiquetées par un nom de message. L'écoulement du temps sur chaque processus est symbolisé par un trait vertical et les messages par des flèches de l'émetteur vers le récepteur. De nombreuses publications contiennent ces représentations, qui servent à illustrer par un exemple le comportement d'un système (Hélouët et Caillaud, 2001).

2.1. Les bMSC

Un diagramme de base (bMSC) est une structure (E, L, I, λ, <, tgt) où :

- E est un ensemble d'événements divisé en un ensemble S des événements envoyés et un ensemble R des événements reçus.
- L est un ensemble fini d'étiquettes.
- I est un ensemble fini d'instances.
- λ : E → L * I représente les événements sous la forme d'un couple : λ(*e*) = (l, i). Nous noterons que lbl(e)=l et inst(e)=i si λ(e)= (l, i).
- < est une relation d'ordre partiel sur E telle que pour tout i Є I <i est une relation d'ordre total : <i ⊆ (i(E) * i (E)) où i(E) est le sous-ensemble de E formé des événements associés à l'instance i : i(E) = { e Є E / inst(e)=i }. <i est une relation d'ordre causal entre les événements de l'instance i.
- tgt : S → R est une fonction qui associe l'envoi d'un événement à la réception de cet événement.

Les étiquettes sont nécessaires pour déterminer les types des événements. Le comportement représenté par un bMSC est un ensemble de séquences d'événements déterminées par la priorité causale des événements du bMSC. Les événements sont totalement ordonnés pour chaque instance. Cette relation causale détermine un ordre partiel, noté ≤ , sur les événements de toutes les instances. Ainsi, n'importe quelle séquence d'événements qui respecte cet ordre partiel représente un comportement acceptable du bMSC. Par exemple le comportement du bMSC « Analyser » de la figure 1 comporte une séquence acceptable : Donnée, Commande, Déclencher.

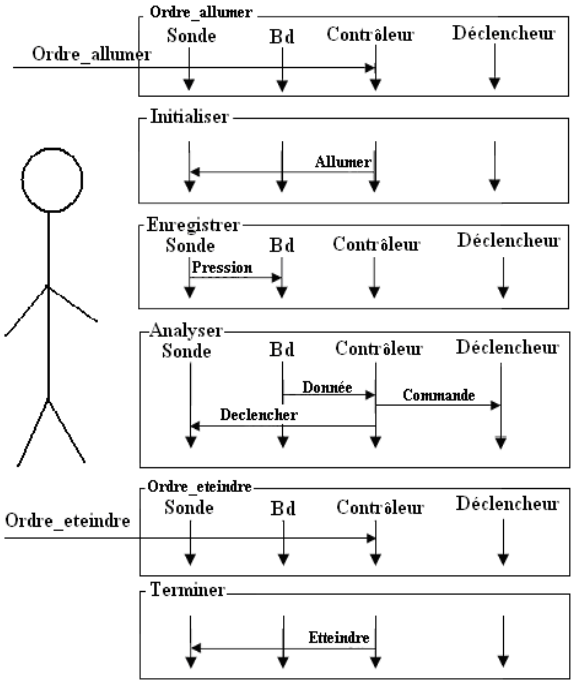

Figure 1. MSC d'une chaudière à vapeur

Soit b = (E, L, I, λ, <, tgt) un bMSC. Un mot l1... l|S| sur l'alphabet L est une linéarisation de b s'il y a un mot s1... s|S| sur l'alphabet S tels que:
- lbl (si) = li pour 1 ≤ i ≤ |S|.
- si ≤ sj alors i ≤ j.

Nous définissons le langage du bMSC *b* comme l'ensemble *L(b)* des linéarisations de b : *L(b) = {w | w* est une linéarisation de *b}*.

2.2. Les hMSC

Les hMSC ont été définis afin de composer des MSC à partir d'autres MSC. Ils peuvent être représentés par un graphe orienté où les nœuds représentent des composants MSC (bMSC ou hMSC) et les arcs indiquent le séquencement entre les composants (voir figure 2). Un triangle permet d'identifier le composant initial.

Un hMSC est une structure (N, A, S0) où:

- N est un ensemble de nœuds.
- A ⊆ (N * N) est un ensemble d'arcs.
- S0 Є N est le nœud initial.

La séquence (éventuellement infinie) de nœuds $w = n0, n1,\ldots$ est un chemin si $ni \in N, n0 = s0$, et $(ni, ni+1) \in A$ pour $0 \leq i < |w|$. On dit qu'un chemin est maximal s'il n'est le préfixe d'aucun autre chemin.

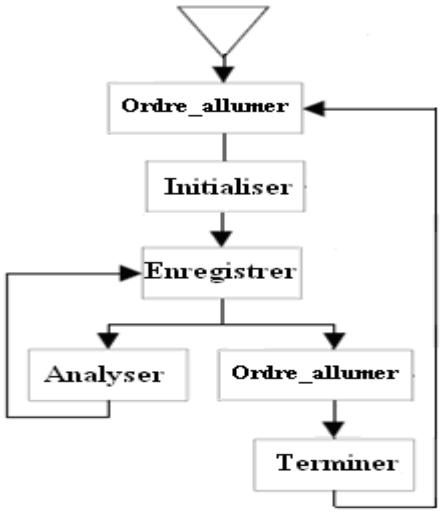

Figure 2. Diagramme hMSC utilisant les composants de la figure 1

Nous pouvons maintenant définir la spécification complète d'un MSC. Lorsque les composants du hMSC sont tous des MSC de base, elle se compose d'un ensemble de bMSC, d'un hMSC et d'une fonction bijective qui associe chaque noeud dans le hMSC à un bMSC. La spécification d'un MSC est une structure (B, H, f) où :

- B est un ensemble de bMSC.
- H= (N, A, s0) est un hMSC.
- f : N→ B est une bijection des nœuds du hMSC vers les bMSCs. Nous noterons α(i) l'ensemble d'étiquettes de messages qui peuvent être reçus ou envoyés par une instance i : α(i) = lbl(i(E)).

Le comportement du MSC est défini par un ensemble de séquences d'étiquettes des messages. Le hMSC ainsi que l'association des nœuds aux bMSC permettent de déduire l'évolution du système d'un scénario à l'autre. Nous supposons que :

- l'envoi d'un message et sa réception sont instantanés,
- les événements d'une même instance respectent l'ordre causal, et
- pour deux composants en séquence dans le hMSC, tous les événements du premier composant se produisent nécessairement avant ceux du second.

Nous définissons la notion de composition séquentielle (ou concaténation) de MSC en respect avec cette dernière hypothèse.

La composition séquentielle de deux bMSC b = (E, L, I, λ, <, tgt) et b' = (E', L', I', λ', <',tgt') est dénotée par (b ● b') et définie par le bMSC (E ∪ E',L ∪ L',I ∪ I', λ ∪ λ', << , tgt ∪ tgt') où << est une relation d'ordre partiel telle que pour i ∈ I <<i est une relation d'ordre total : <<i = <i ∪ <'i ∪ {(max(I (E), min(I (E'))}. Par exemple pour les bMSC (*Enregistrer* • *Analyser*), les séquences possibles d'événements sont : <*Pression, Donnée, Commande, Déclencher*>. Les séquences d'événements déterminées par la spécification du MSC sont celles qui appartiennent au langage de n'importe quelle concaténation maximale des composants MSC autorisée par le hMSC.

Soit Spéc = (B, H, f) la spécification d'un MSC. On dit que b est un bMSC maximal dans Spéc s'il y a un chemin maximal *n1, n2,..., nk* dans H tels que b = f (n1) • f (n2) • … f (nk)

On définit le langage de Spéc comme l'ensemble *L(Spéc)* des bMSC maximaux de Spéc, où *L(Spéc) = {w ∈ L(b) | b* est un bMSC maximal de Spéc*}*.

3. DES MSC AUX DEVS

De nombreuses approches de génie logiciel permettent d'induire des machines à états finis à partir de modèles de scénarios (Liang et al. 2006). D'autres approches proposent de construire des modèles DEVS à partir de certains diagrammes UML (Risco-Martin et al. 2007), tels que les diagrammes de séquence (KeungSik et al. 2006) qui représentent des scénarios. Kruger (Kruger, 2000) a proposé une démarche de création des automates à états finis à partir d'un ensemble de bMSC. La procédure de transformation se compose de cinq phases successives :

- Vérification : Cette phase consiste à vérifier que l'ensemble des comportements décrits par chaque bMSC est un ensemble de suites d'événements respectant l'ordre causal.
- Projection: Après avoir choisi l'instance pour laquelle il est prévu de construire un automate, chacun des bMSC est projeté sur cette instance, ce qui consiste à enlever les événements auxquels l'instance choisie ne participe pas et à supprimer les scénarios vides. Ceci réduit le nombre et la taille des scénarios.
- Normalisation: Il s'agit d'identifier les événements qui permettront de déterminer les phases initiales et finales des automates.
- Transformation de chaque bMSC en un automate : chaque événement correspond à une transition dans l'automate et des états intermédiaires sont ajoutés pour lier les transitions.
- Optimisation: Les automates résultants sont essentiellement des machines à états finis non déterministes, un des algorithmes d'optimisation bien connus de la théorie des automates peut alors être appliqué pour réduire au minimum le nombre d'états et de transitions des automates résultats.

Le comportement du système est finalement défini comme la composition parallèle de tous les automates obtenus. En d'autres termes, le comportement du système est le résultat de l'exécution d'une manière asynchrone des

composants mais en se synchronisant sur tous les messages partagés.
Notre approche doit nous permettre d'obtenir des modèles DEVS à partir d'un ensemble de bMSC tel que les relations entre les bMSC sont décrites par un hMSC. Comme dans la démarche de Kruger, nous allons construire un modèle pour chaque instance mais en considérant tous les bMSC à la fois. Le comportement final sera décrit grâce au couplage des modèles obtenus pour chaque instance.

3.1. Le formalisme DEVS

Conçu par Zeigler pour permettre une modélisation rigoureuse à événements discrets (Zeigler, 2000), le formalisme DEVS introduit la possibilité d'évolution autonome du modèle grâce à la durée de vie des états et à la fonction de transition interne. Il permet d'effectuer la description comportementale des systèmes à deux niveaux. D'une part, un modèle DEVS atomique décrit le comportement du système par une séquence de transitions déterministes entre des états séquentiels, la manière dont le système réagit à des événements externes et comment il génère des sorties. D'autre part, à un niveau d'abstraction supérieur, un modèle DEVS couplé décrit le système en tant que réseau de composants, qui sont soit des modèles DEVS atomiques soit des modèles DEVS couplés. Pour tout modèle DEVS couplé, un DEVS atomique équivalent peut être formé. Le formalisme DEVS permet la modélisation de systèmes complexes, la possibilité d'une modélisation modulaire et hiérarchisée permettant de modéliser de manière réaliste les comportements. De plus, la possibilité de couplage entre modèles DEVS permet l'intégration de différents composants.

3.1.1. Modèle DEVS atomique

Un modèle DEVS atomique (Zeigler, 2000) est défini par la structure suivante : < *X, Y, S, δext, δint, λ, ta* >
telle que les entrées X et les sorties Y du modèle sont basées sur la notion d'événement.

- X = {(*p,v*) | *p* Є IPorts; *v* Є VX} est l'ensemble des ports et des valeurs d'entrée, avec IPorts l'ensemble des noms des ports d'entrée et VX est l'ensemble des valeurs possibles sur les ports d'entrée.
- Y = {(*p,v*) | *p* Є OPorts; *v* Є VY} est l'ensemble des ports et des valeurs de sortie, avec OPorts l'ensemble des noms des ports de sortie et VY est l'ensemble des valeurs possibles sur les ports de sortie.
- S est l'ensemble des états du système.
- δext est la fonction de transition externe, elle exprime l'évolution du modèle sur occurrence des événements externes.
- δint est la fonction de transition interne, elle exprime l'évolution autonome du modèle
- λ est la fonction de sortie du système.
- ta est la fonction d'avancement du temps.

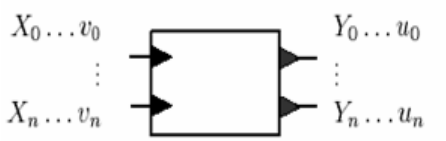

Figure 3. Représentation d'un modèle DEVS atomique

Les notions de port d'entrée et de port de sortie permettent de représenter graphiquement une vue externe d'un modèle DEVS par une boîte où figurent ces ports (figure 3). Les vecteurs d'entrée et de sortie sont l'union de tous les ports du modèle. Les ports d'entrée figurent sur les triangles à l'intérieur de la boîte, et les ports de sortie sur les triangles à l'extérieur de la boîte.

3.1.2. Modèle DEVS couplé

Un modèle couplé (Zeigler, 2000) est formé en connectant plusieurs modèles DEVS (atomiques ou couplés) entre eux. Un tel modèle est défini par :

- l'ensemble des modèles qui le composent,
- l'ensemble des ports d'entrée qui recevront les événements externes,
- l'ensemble des ports de sortie qui émettront les événements,
- les couplages entre ports d'entrée et ports de sortie des modèles composant le modèle couplé.

Il est ainsi défini formellement par la structure suivante : < *X, Y, M, EIC, EOC, IC* > avec

- X = {(p, v) | p Є IPorts, v Є V} l'ensemble des couples port-valeur d'entrée.
- Y = {(p, v) | p Є OPorts, v Є V} l'ensemble des couples port-valeur de sortie.
- M l'ensemble des composants DEVS
- EIC l'ensemble des connections en entrée.
- EOC l'ensemble des connections en sortie.
- IC l'ensemble des connections internes.

Par exemple, la figure 4 représente un modèle couplé N composé de deux modèle atomiques A et B, tel que les différentes connections sont formellement définies par :

- IPorts = {in}
- OPorts = {out1, out2}
- EIC = {((N, in), (A, in1))}
- EOC = {((B, out), (N, out1)), ((B, out), (N, out2)), ((A, out), (N, out1))}
- IC = {((A, out), (B, in)), ((B, out), (A, in2))}

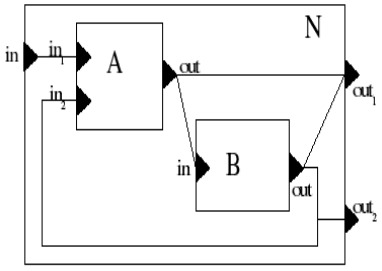

Figure 4. Représentation d'un modèle DEVS couplé

3.2. Traduction des MSC en DEVS

L'algorithme de la figure 5 peut être employé pour construire un modèle DEVS atomique pour chaque instance mentionnée dans la spécification du MSC. En outre, si les modèles DEVS atomiques de toutes les instances peuvent communiquer entre eux, alors nous obtiendrons une exécution des spécifications du MSC. Pour cela, on construit un modèle couplé final qui est composé de tous les modèles atomiques obtenus pour chaque instance du système de telle façon que la sortie d'un modèle atomique soit l'entrée d'un autre. Le modèle couplé décrit le comportement global du système.

Dans notre approche, nous supposons que les relations d'ordre de chaque bMSC impliqué dans le hMSC sont totales de façon à éviter les situations de blocage (voir figure 15), et qu'une séquence de deux bMSC dans le hMSC a la signification donnée au paragraphe 2.2.

L'algorithme synthétise une instance à la fois. Nous l'expliquerons brièvement en l'appliquant sur l'instance *Contrôleur* du hMSC de la figure 2.

En donnant les spécifications d'un MSC et une instance O, l'algorithme traite les événements dont la projection sur la ligne de vie de O est positive sur chaque bMSC et génère un modèle DEVS associé à O. La projection πO(S) d'un diagramme bMSC S sur un objet O est la restriction de la relation d'ordre partiel < aux événements situés sur la ligne de vie de l'objet O. Nous allons considérer la projection comme un mot πO = (*e1.e2...en)* tel que φ(O) = {*e1, . . . en*} et ainsi *e1* < *e2* < . . . < *en*. Par exemple, le mot « Donnée, Commande, Déclencher » est la projection du bMSC « Analyser » sur l'objet « Contrôleur ». Chaque modèle DEVS est en quelque sorte le comportement partiel de l'objet dans le hMSC. Les réceptions des messages deviennent des transitions externes dans les modèles DEVS et les émissions deviennent des transitions internes. Notons que les transitions externes en DEVS sont représentées par des arcs libellés avec le nom du port d'entrée suivi du symbole « *?* » et la valeur de l'événement déclenchant le changement d'état. Et les transitions internes sont représentées par des arcs en traits pointillés. La fonction de sortie figurera sur l'arc suivie du symbole « *!* » et la valeur de l'événement produit.

3.3. Algorithme (en pseudo-code)

```
Procédure Synthèse (Specification Spec{}, object O)
{
    X :={};Y :={}; δint() ; δext() ; S:={}
    Temp_devs:={}{};

    // On désigne l'ensemble des successeurs pour
    chaque bMSC du hMSC. Si la projection de
    l'instance du système sur le bMSC successeur est
    nulle alors le successeur du bMSC devient le
    successeur du bMSC dont la projection est nulle,
    et ainsi de suite.

    Nb_scenarios:=spec.longueur();
    Pour  k de 1 à Nb_scenarios {
      Pour i de 1 à N { //N nombre des  bMSC
        bi:= Spec{k}.getbMSC(i);
        Tantque  ( πO(next(bi){0}) == 0 ) et
              ( next(bi){j} <> null ) {
          next(bi):= next(bi) -{next(bi){0}};
        }}}

    // On construit pour chaque bMSC le modèle
    DEVS équivalent.
    Pour  k de 1 à Nb_scenarios {
      Pour i de 1 à N{ //N nombre des bMSC
        bi:= Spec{k}.getbMSC(i);
        Si ( πO (bi)<>0 )
            Di:=Creat_Devs(bi,O, currentState) ;
        Sinon Di:=null;
        Temp_devs{k} := Temp_devs {k}+ {Di};
      }}

    // On relie chaque DEVS équivalent à un bMSC
    au DEVS équivalent au bMSC successeur.
    Pour  k de 1 à Nb_scenarios {
      Pour  i de 1 à  N {
        bi:= Spec(k).getbMSC(i);
        Dbi:= devs_equiv(bi);
        Si (πO (bi)<>0) {
          Pour  j de 0 à next(bi).longueur()-1 {
            Dnexti:=devs_equiv(next(bi));
            Dbi.S{S.longueur()}:=Dnexti.S{1});
          }}}}

Procédure Creat_DEVS (bMSC M, object O, var
state s) {
  Devs D = (X{}, Y{}, S{}, δint, δext , λ, ta) ;
  ProjectedEvents: = πO (M);
    Create_state (s) ;
    currentState :=s ;
    S = S +{s} ;
    Pour i de 1 à (ProjectedEvents) {
          ei := ProjectedEvents[i] ;
          v := ei.value ;
          p:=ei.port;
          Create_state (s+1) ;
          S = S + {s+1};
          Si (ei) est un événement d'entrée {
            δext (currentState, ei, IN ?v) := s+1 ;
            ta (currentState) := infinite;
            X=X+ {(p, v) | p ∈ IPorts; v ∈ Vx} ;
          }
          Sinon Si (ei) est un événement de sortie {
            δint(s+1) := currentState ;
            ta (currentState) :=0;
            λ (s+1) := (p !v) ;
            Y =Y+ {(p; v) | p ∈ OPorts; v ∈ Vy} ;
          }
          currentState := s+1 ;
    }
    s := currentstate+1 ;
    Return D ;
}
```

Procédure Fusion de tous les modèles atomiques en un seul modèle atomique. Cette étape induit le modèle DEVS représentant le comportement global du système pour chaque instance, depuis les précédents modèles associés à chaque scénario. Pour cela, nous avons fusionné les états qui reçoivent les mêmes événements à partir de même état. Alors, pour tous les modèles atomiques obtenus pour chaque scénario:

- Cas d'une transition interne : si δint (Si) =Sj avec λ (Si)= (p!v) et δint (Si) = S'j avec λ' (Si) = (p !v) talors Sj:=S'j;
- Cas d'une transition externe : isi δext (Si, e, p ?v) = Sj et δext (Si, e, p ?v)= S'j alors Sj:=S'j;

Figure 5. Algorithme de traduction de chaque instance de plusieurs hMSC en un modèle DEVS atomique

Remarquons que l'algorithme ajoute des états intermédiaires pour lier les transitions. Par exemple, en considérant que l'état récent est *qi*, pour tout *i (1 ≤ i ≤ n)*, si un événement *e* produit le changement d'états *(qi → q'i)* alors *qi* est remplacée par *q'i*, et *qi* est abandonné, autrement dit l'état récent est *qi'*.

En associant une transition externe à chaque événement en entrée, et une transition interne à chaque événement de sortie, on obtient pour l'instance *« Contrôleur »* du hMSC de la chaudière à vapeur :

• Ordre_allumer : in!ordre_alumer.
δint(Init) = {}; δext(Init) = {in? ordre_allumer};
S(Init) = {s0, s1};
• Initialiser: out!allumer.
δint(Init) = {out!allumer}; δext(Init) = {};
S(Init) = {s1, s2};
• Enregistrer: _____.
• Analyser: in?donnée->out!commande> out!declencher
δint(Anal) = {out!commande, out!declencher};
δext(Anal) = {in?donnée};
S(Anal) = {s3,s4,s5};
• Ordre_eteindre : in ?ordre_eteindre
δint(Term) = {}; δext(Term) = {in? ordre_eteindre};
S(Term) = {s2, s5};
• Terminer: out!eteindre.
δint(Term) = {out!eteindre}; δext(Term) = {};
S(Term) = {s5, s0};

Sachant que le chemin du hMSC est le suivant :
Next (Init) = Ordre_allumer
Next (Ordre_allumer) = Initialiser
Next (Initialiser) = Enregistrer.
Next (Enregistrer) = Analyser, Ordre_eteindre.
Next (Analyser) = Enregistrer.
Next (Ordre_eteindre) = Terminer.
Next (Terminer) = Initialiser.
Comme (πContrôleur(Enregistrer)=0), la projection du bMSC *Enregistrer* sur l'instance *« Contrôleur »* est nulle, le chemin de l'inastance dans le hMSC de la figure 2 devient :

• Debut (*Contrôleur*) =Ordre_allumer.
• Ordre_allumer = (in?ordre_allumer -> Initialiser).
• Initialiser = (out!allumer -> (hiddenAction -> Analyser | hiddenAction -> Terminer)).
• Analyser = (in?donnée -> out!commande -> out ! declencher -> (hiddenAction -> Analyser | hiddenAction -> Ordre_eteindre)).
• Ordre_eteindre =(in?ordre_eteindre -> terminer).
• Terminer = (out!eteindre -> Initialiser).

En reliant les états de chaque modèle atomique partiel correspondant à un bMSC dans le hMSC, on obtient le diagramme DEVS < X, Y, S, δext, δint, λ, ta > où :

• S = S∪S(All)∪S(init)∪S(anal)∪S(Et)∪S(term)
• δext=δext∪δext(All)∪δext(init)∪δext(anal)∪ δext(Et) ∪δext(term)
• δint = δint∪δint(all)∪δint(init)∪δin(anal)∪ δin(Et) ∪δin(term)
• X=X∪X(All)∪X(init)∪X(anal)∪X(Et)∪ X(term)
• Y=Y∪Y(All)∪Y(init)∪Y(anal)∪Y(Et)∪ Y(term)

Par exemple, en se basant sur le chemin du hMSC précédent, on obtient pour l'instance *« Contrôleur »* le modèle DEVS atomique de la figure 6). Dans cette figure, les états en pointillés sont ceux d'où part un arc en pointillé (transition interne), des états en trait plein partent les arcs en trait plein correspondant aux transitions externes.

Les figures 6 à 9 représentent les modèles DEVS atomiques obtenus pour chaque instance du système par l'algorithme ci-dessus.

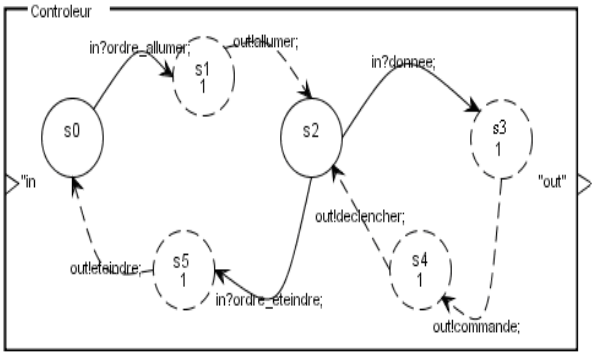

Figure 6. Diagramme DEVS de l'instance Contrôleur

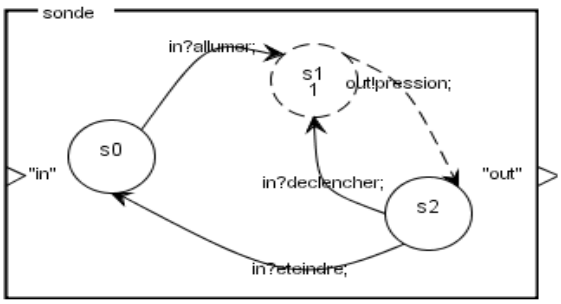

Figure 7. Diagramme DEVS de l'instance Sonde

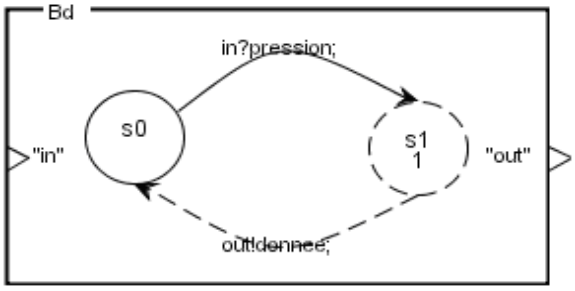

Figure 8. Diagramme DEVS de l'instance Bd

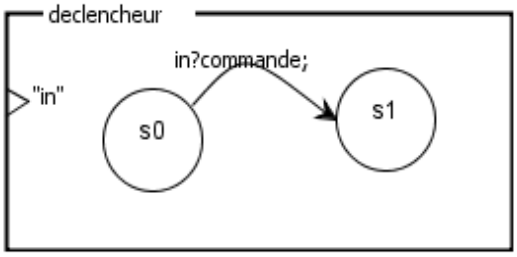

Figure 9. Diagramme DEVS de l'instance Déclencheur

3.4. Scénarios multiples

L'algorithme peut aussi traiter plusieurs scénarios à la fois pour la même instance du système, décrits dans des hMSC différents. Ajoutons pour cela le scénario « Erreur » aux comportements de la chaudière à vapeur. Lorsque le contrôleur reçoit l'ordre d'allumer, celui-ci allume la sonde. Si par la suite il reçoit une erreur extérieure alors le contrôleur éteint la sonde. Le diagramme MSC Erreur est décrit par les figures 10 et 11.

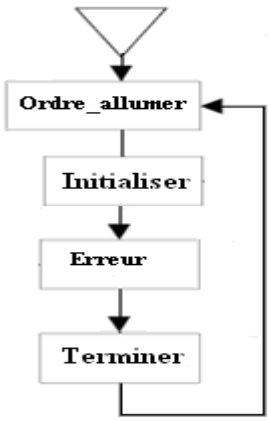

Figure 10. Le hMSC décrivant le scénario Erreur

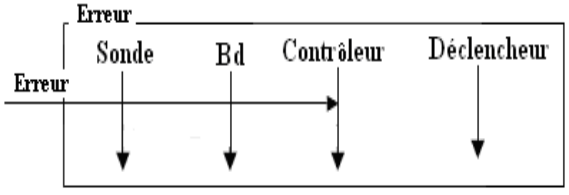

Figure 11. Le bMSC Erreur

Le diagramme DEVS équivalent à l'instance « Contrôleur » devient celui de la figure 12.

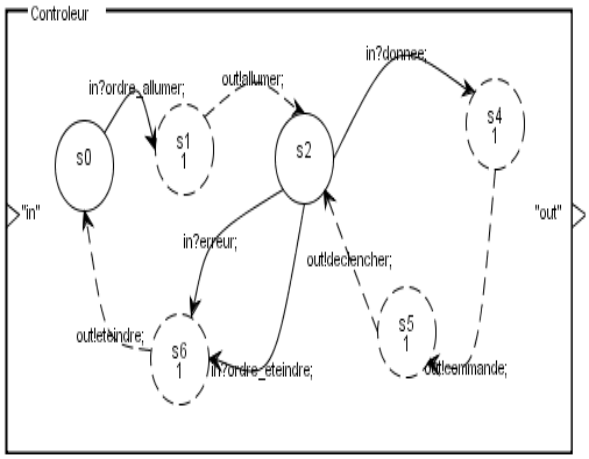

Figure 12. Diagramme DEVS de l'instance Contrôleur

3.5. Modèle DEVS couplé final

Prenons notre exemple de la chaudière à vapeur, en parcourant les instances du système l'une après l'autre, on peut effectuer la connexion entre les modèles DEVS atomiques obtenus pour chaque instance. en reliant les entrées d'un modèle avec les sorties de l'autre. Le diagramme DEVS couplé obtenu est donné sur la figure 13.

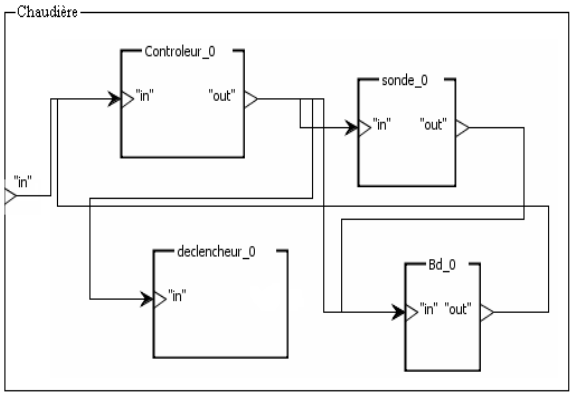

Figure 13. Diagramme DEVS couplé final

4. ANALYSE

Il s'agit d'étudier la relation entre le comportement spécifié dans les scénarios et celui obtenu dans le modèle DEVS synthétisé. Quatre situations principales peuvent se produire qui sont représentées en figure 14.

- Égalité entre les deux comportements : Les comportements spécifiés dans les scénarios (MSC) sont exactement les mêmes que ceux obtenus dans le modèle DEVS. Il est évident que ce cas est la situation idéale. Dans ce cas, la synthèse est une étape de transition sûre et les résultats de la synthèse peuvent être utilisés pour la vérification et pour la simulation sans risque d'incohérence.

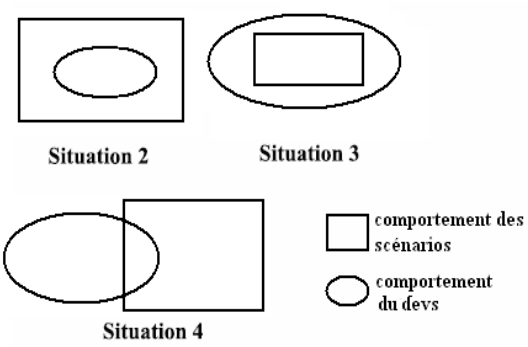

Figure 14. Relations entre le comportement des scénarios et celui de DEVS.

- Inclusion sens 1: Les comportements du modèle DEVS sont inclus dans les comportements des scénarios. Cette situation se produit en cas de blocage comme sur la figure 15, où la précédence entre les événements e3 et e5 par exemple n'est pas explicite. Notons que ceci pourrait être évité en séparant le bMSC2 en deux composants. L'algorithme de construction a choisi aléatoirement une précédence entre les événements et la simulation pourra conduire à un blocage si les entrées ne respectent pas la précédence choisie.
- Inclusion sens 2: Les comportements spécifiés dans les scénarios pourraient être inclus dans ceux du modèle DEVS. Dans cette situation, la synthèse générerait des comportements supplémentaires. Toutefois, dans les conditions d'application de cet algorithme cette situation ne peut pas se produire
- Incomparable : La dernière situation ne peut se produire lorsque la situation précédente n'est pas envisageable.

4.1. Exemple de blocage

Considérons pour cet exemple que l'on souhaite synthétiser toutes les instances à la fois du système (vue globale). Le MSC de la figure 15 se compose de quatre instances, appelées <'I1', 'I2','I3' et 'I4'>, et trois messages, 'm1', 'm2' et 'm3'. L'échange du message 'm1' doit être accompli avant que le message 'm2' puisse être envoyé parce qu'ils sont temporellement ordonnés dans la même instance.

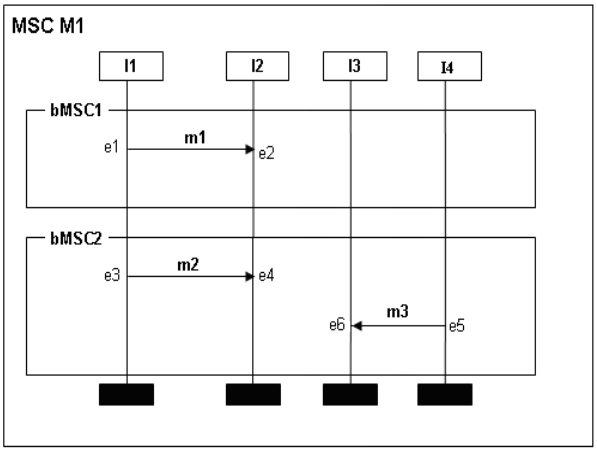

Figure 15. Exemple d'un MSC avec blocage

Dans le bMSC2, l'événement e3 (envoi du message 'm2') et l'événement e5 (envoi du message 'm3') ne sont pas ordonnés sur la même instance et on ne peut pas savoir quel événement se déclenchera avant l'autre (concurrence entre les événements e3 et e5).

5. VALIDATION

Pour la validation de la spécification du comportement du système obtenu, on a décidé de simuler le modèle résultat. Pour cela, on a utilisé l'outil LSIS_DME d'édition et de simulation de modèles DEVS, développé au sein de notre laboratoire. Un des avantages de cet outil est qu'il dispose d'une interface de description textuelle des modèles DEVS au format XML ce qui évite de traduire le modèle dans le langage approprié du simulateur. Pour les travaux futurs on utilisera soit la génération automatique de test, soit la vérification formelle ou model-checking qui est basé principalement sur des logiques temporelles. Les propriétés que le modèle doit vérifier sont exprimées sous forme de propositions logiques et l'utilisation d'un model-checker permet de prouver que le modèle est correct vis-à vis de ces propriétés, ou d'exhiber une trace d'exécution non satisfaisante dans le cas où le modèle est incorrect. Les résultats obtenus pendant la simulation sont les suivants :

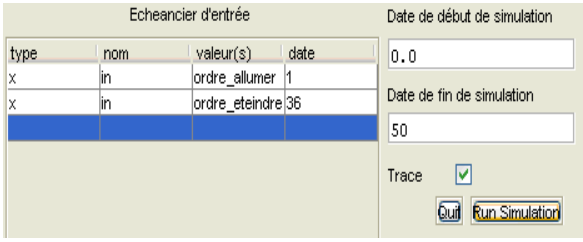

Figure 16. Remplissage des échéanciers d'entrée

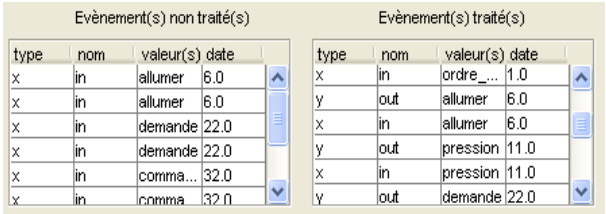

Figure 17. Les événements traités et non traités

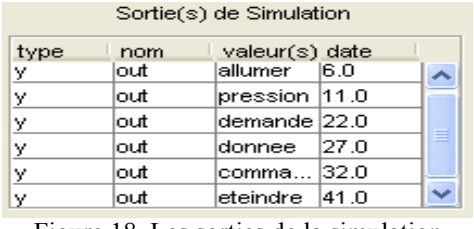

Figure 18. Les sorties de la simulation

On remarque d'après la simulation que les comportements obtenus sont exactement les mêmes que ceux spécifiés dans le MSC de la chaudière à vapeur.

6. CONCLUSION

Dans notre article, nous avons présenté une méthode de synthèse de scénarios permettant de passer à une modélisation rigoureuse de systèmes à événements discrets en générant une spécification comportementale à partir d'un ensemble de scénarios. Notre principal objectif est de faciliter la tâche de modélisation pour le client/utilisateur afin de satisfaire son besoin. Pour les systèmes complexes, il est toujours plus aisé de décrire plusieurs exemples d'exécution que de déduire le comportement global du système. La synthèse en modèles DEVS permet en outre de valider le comportement obtenu par simulation.

Nous avons utilisé les MSC (Message Sequence Charts) comme langage de description des scénarios. Ce langage relativement simple et intuitif est basé sur une syntaxe textuelle, munie d'une sémantique formelle exprimée sous forme d'algèbre de processus. De plus, la représentation graphique qui lui est associée le rend facilement compréhensible par les utilisateurs non familiers des méthodes formelles.

L'algorithme de synthèse respecte l'ordre causal entre les événements de la même instance. La spécification comportementale du système DEVS obtenu, correspond en l'absence de blocage aux exigences exprimées dans les scénarios (MSC).

D'autres approches comme celle de (Damas et al. 2006) peuvent être envisagées qui s'appliquent directement à des bMSC de façon à libérer l'utilisateur de l'identification de certains concepts plus abstraits comme les répétitions. L'identification des boucles dans le comportement global du système est alors reportée à la phase de synthèse. Les automates induits à partir des scénarios décrivent les comportements représentés par les scénarios mais aussi nécessairement des comportements supplémentaires. Ces approches sont basées sur la construction de l'arbre accepteur des préfixes à partir de toutes les linéarisations des bMSC, et s'inspirent des travaux de recherche sur l'inférence grammaticale ou la programmation automatique. Elles se différencient essentiellement par les techniques de regroupement des états.

REFERENCES

Brian L., Hans E., 2004. UML 2 toolkit. *Whiley Publishing.* OMG press.

Damas C., Lambeau B., van Lamsweerde A., 2006 Scenarios, Goals, and State Machines: a Win-Win Partnershi for Model Synthesis, 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering, Portland, Oregon, USA, pp. 197-207.

Damm W., Harel D., 1999. *LSC: Breathing Life into Message Sequence Charts.* FMOODS'99 IFIP TC6/WG6.1 Third International Conference on Formal Methods for Open Object-Based Distributed Systems.

Hélouët L., Caillaud B., 2001. An event Structure Semantics for Message Sequence Charts. *Mathematical Structures in Computer Science*.vol. to appear.

Liang H., Dingel J., Diskin Z., 2006, *A comparative Survey of Scenario-based to State-based Model Synthesis Approaches*, SCESM'06 : International Workshop on Scenarios and State Machines: Models, Algorithms, and Tool, Shangaï, China, pp. 5-12, May.

ITU., 1996. Z.120 ITU-TS. Recommendation Z.120. *Message Sequence Chart (MSC). Technical report, ITU-TS*, Genova.

KeungSik C., SungChul J., HyunJung K., Doo-Hwan B., DongHun L., 2006, UML-based Modeling and Simulation Method for Mission-Critical Real-Time Embedded System Development, IASTED Conference on Software Engineering 2006, 160-165.

Krüger H., 2000 *Distributed System Design with Message Sequence Charts.* Institut für Informatik, Ludwig-Maximilians-Universität München

Reniers M., 1995. *Static semantics of Message Sequence Charts.* In SDL'95 with MSC in CASE. Proceedings of the Seventh SDL Forum. Oslo. Elsevier Science Publishers B.V.

Risco-Martín, J.L., Mittal, S., Zeigler, B.P., 2007, *From UML Statecharts to DEVS State Machines Using XML*, Multi-paradigm Modeling, IEEE/ACM International Conference on Modeling Languages and System, Nashville, Sept.

Zeigler B., Kim T., Praehofer H., 2000, Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems, Academic Press.