Vers un environnement de création de services fondé sur les scénarios enrichis[1]

Aziz Salah* — Rachida Dssouli* — Guy Lapalme* — Daniel Vincent**

* *Département d'Informatique et de Recherche Opérationnelle,*
Université de Montréal.
C.P.6128, Succ Centre-ville Montréal, P.Q. Canada H3C 3J7
{salah,dssouli,lapalme}@iro.umontreal.ca

** *France Télécom - Branche Développement*
Centre National d'Études des Télécommunications, FT.CNET/DTL/MSV
Technopole Anticipa - 2, avenue Pierre Marzin - 22307 Lannion Cedex - France
daniel.vincent@cnet.francetelecom.fr

RÉSUMÉ. *Les scénarios constituent une technique attrayante de plus en plus utilisée pour la spécification des systèmes. Cet article décrit une approche basée sur les scénarios enrichis pour la création de services. L'approche proposée permet la génération automatique d'une spécification à partir des scénarios d'utilisation. Les scénarios sont représentés sous forme de Message Sequence Charts qui sont intégrés dans un automate temporisé étendu. Dans notre approche, les scénarios décrivent les contraintes temporelles et les contraintes sur les paramètres qui contrôlent le comportement du système.*

ABSTRACT. *Scenarios offer an attractive technique for system specification. This paper describes a service creation approach based on scenarios. The approach supports automatic system specification synthesis from use case scenarios. Scenarios are expressed in Message Sequence Charts which are integrated within an extended timed automaton. Our approach deals with scenarios describing system behavior controlled by time constraints and system variable constraints.*

MOTS-CLÉS : *spécification formelle, intégration des scénarios, création de service, MSC, génération automatique de spécifications*

KEYWORDS: *formal specification, enriched scenarios integration, service creation, MSC, automatic specification synthesis*

1. Ce travail est réalisé dans le cadre d'un contrat de recherche externe entre France Télécom et le groupe de téléinfomatique de l'Université de Montréal

1. Introduction

Durant cette décennie, l'expérience a montré que l'approche scénario constitue un moyen efficace pour étudier le comportement d'un système dans une situation particulière. Une fois ce comportement partiel approuvé, il faut l'intégrer dans le comportement global afin de s'assurer du bon fonctionnement du système en entier. L'approche scénario est ainsi vue comme le "diviser pour régner" de l'activité de spécification. En industrie, l'approche scénario est utilisée à différents niveaux du cycle de vie d'un système, par exemple; dans les méthodes de conception orientées objet récemment unifiées en UML [RAT 97], pour la définition des interactions Homme-Machine [NAR 92], pour la compréhension des exigences informelles [CAR 92], et pour la génération d'une spécification formelle d'un système [SOM 97]. Dans cet article nous proposons une nouvelle méthode incrémentale et automatique pour la génération d'une spécification formelle d'un système à partir des scénarios d'utilisation. Cette méthode s'intègre dans le cadre d'un environnement de création de services qui permet de synthétiser une spécification formelle cohérente et aussi complète que possible. Notre méthode se distingue par la prise en compte du temps et des variables dans le comportement spécifié par les scénarios. L'utilisation de variables permet de factoriser plusieurs scénarios en un seul. Les spécifications générées sont exprimées dans le modèle des automates temporisés étendus. Par ailleurs, l'utilisation des MSC (*message sequence chart*) (Z.120) pour la représentation des scénarios rend notre approche facilement utilisable par la communauté croissante utilisant ce formalisme.

La section 2 présente une revue des travaux dans le domaine de la génération de spécifications à partir des scénarios. La section 3 est dédiée aux fondements théoriques de notre approche. La section 4 présente les différentes activités de notre environnement de création de services et soulève les problèmes posés ainsi que les choix effectués. Nous concluons en présentant des directions pour les travaux ultérieurs.

2. Problématique

Il est reconnu que l'activité de spécification constitue une phase critique et déterminante dans le cycle de vie d'un système. En effet, les erreurs commises pendant cette activité sont difficiles à détecter et coûteuses à corriger à la fin du cycle de développement. L'utilisation des langages formels a permis d'éliminer certaines erreurs dues aux ambiguïtés des expressions informelles. Mais cela reste insuffisant. Nous croyons que l'automatisation de l'activité de spécification permet d'éliminer certaines erreurs humaines et de systématiser le processus. L'approche scénario permet d'atteindre cet objectif en synthétisant une spécification formelle par intégration des scénarios d'utilisation. Peu de travaux ont abordé ce problème et très peu présentent des approches complètement automatiques. Nous comparons ces approches sur deux critères:

– sur leur niveau d'abstraction et de détails véhiculés dans les scénarios tels que les paramètres, les variables et le temps

– sur leurs méthodes d'intégration des scénarios explicite ou implicite.

La méthode explicite fournit un opérateur indiquant l'enchaînement des scénarios alors que dans l'implicite il est déduit par l'algorithme. Nous présentons maintenant quelques travaux reliés aux nôtres et nous définissons la problématique à résoudre dans cet article.

Glinz [GLI 95] représente ses scénarios sous forme de statechart [HAR 87]. Les scénarios sont intégrés d'une manière explicite sous forme de patrons imposant un ordre d'exécution qui va se retrouver dans la spécification générée. Cette approche a l'avantage de traiter des scénarios concurrentiels par entrelacement grâce au formalisme des statecharts. Par contre, après leur intégration, les scénarios restent visibles dans le statechart de la spécification globale. Ceci suppose que les scénarios sont disjoints et sans chevauchement.

Koskimies et al. [KOS 94] proposent un algorithme pour la génération d'une machine à états décrivant le comportement dynamique d'un objet dans la méthode OMT à partir de diagrammes d'interactions aussi appelés scénarios ou “use cases”. La génération de la machine à états se fait par induction en se basant sur l'algorithme de Biermann [BIE 76] qui est sensible à l'ordre d'entrée des traces mais qui garantit toujours une machine à états déterministe minimale. À cause du manque de sémantique dans les actions, les scénarios risquent d'être combinés d'une manière indésirable.

L'approche de Khriss et al. [KHR 98], comme [KOS 94], est utilisée dans une méthode de conception orientée objet. De même [KHR 98] partage avec [GLI 95] l'utilisation des statecharts mais elle se distingue par la possibilité de traiter les chevauchements entre scénarios. Par contre, l'approche nécessite des interventions manuelles pour l'étiquetage des états.

Par ailleurs, dans l'approche de Somé et al. [SOM 95, SOM 97], les scénarios sont représentés sous forme de traces temporisées exprimées en langage naturel restreint. Les scénarios capturés sont intégrés de manière implicite sous forme d'un automate temporisé. Cette approche permet aussi la vérification de la spécification générée.

D'autres approches ont été proposées pour la génération de spécifications à partir de scénarios et utilisant d'autres formalismes tels que les réseaux de Pétri [DAN 97] et les grammaires régulières [HSI 94]. Mais, à notre connaissance, les approches proposées pour la génération automatique des scénarios ne permettent pas l'intégration de scénarios enrichis à la fois de contraintes temporelles et de contraintes sur les variables. Ceci constitue l'une des principales contributions de notre approche. Ainsi nous proposons une nouvelle sémantique formelle qui permet l'intégration automatique des scénarios enrichis. En plus d'être contraints par le temps, les événements dans ce type de scénarios sont paramétrés. Certains paramètres sont des variables libres dans un domaine donné. Les problèmes que pose l'intégration de tels scénarios sont les suivants: Quel est le modèle approprié pour les représenter en vue de leur intégration? Comment formaliser l'état du système pour faciliter cette intégration? Comment traiter le chevauchement de comportements entre les scénarios? Comment factoriser ce chevauchement dans la spécification générée? Quelles sont les contraintes à ajouter ou à propager afin de préserver dans la spécification les comportements décrits par les scénarios?

3. Fondement théorique de notre approche

Dans notre approche, nous utilisons le modèle des automates temporisés étendus par les variables (abrégé par ETA pour *E*xtended *T*imed *A*utomata) qui est un modèle hybride entre les modèles des automates temporisés [ALU 94] et des machines à états finis étendues. Le modèle des ETA permet de représenter le contrôle temporisé tout en mettant en évidence le flux de données.

Définition 1. *Un ETA est un sixtuplet* $A = (M_A, S_A, T_A, s^o_A, H_A, V_A)$ *où*

– M_A *est un alphabet fini. avec* $M_A = I_A \cup O_A$ *et* $I_A \cap O_A = \emptyset$ *où* I_A *est l'ensemble des entrées et* O_A *l'ensemble des sorties,*
– S_A *est l'ensemble d'états,*
– T_A *est l'ensemble des transitions,*
– s^o_A *est l'ensemble des états initiaux,*
– H_A *est un ensemble fini d'horloges et*
– V_A *est un ensemble fini de variables*

Chaque transition $t \in T_A$ *est représentée par un sixtuplet* $t = (s, \delta, \gamma, m, \lambda, s')$*, où* $s \in S_A$ *est l'état de départ de la transition,* $s' \in S_A$ *est son état d'arrivée,* $m \in M_A$ *est un symbole de l'alphabet,* δ *est un ensemble de contraintes sur les valeurs des horloges à satisfaire avant le franchissement de la transition,* γ *est un ensemble de contraintes sur les valeurs des variables vérifiée avant le franchissement de la transition,* $\lambda \subset H$ *est un ensemble d'horloges à réinitialiser après le franchissement de la transition .*

Définition 2. *Chaque contrainte d'horloges* $\phi \in \delta$ *est une formule respectant la grammaire suivante:*$\phi ::= x \sim c \mid \phi \wedge \phi \mid \phi \vee \phi \mid True$*, avec* $\sim \in \{\leq, <, =, \geq, >\}$*,* x *une horloge de* H *et* c *une constante numérique dans* $\mathbb{Q}$*.*

Définition 3. *Chaque contrainte sur variable* $\psi \in \gamma$ *vérifie la grammaire suivante:* $\psi ::= \nu \mathcal{R} a \mid \psi \wedge \psi \mid \psi \vee \psi \mid True$*, où* $\mathcal{R}$ *est une relation définie pour le type de la variable* ν *et où* a *est l'une de ses valeurs plausibles.*

Par exemple pour une variable de type entier, $\mathcal{R}$ peut être l'une des relation $=, \leq, \geq, <$ ou $>$ et a est une valeur entière.

L'interprétation donnée à la transition t est que lorsque l'automate A est dans l'état s, et les contraintes δ et γ sont satisfaites, l'événement m est produit, ensuite les horloges dans λ sont réinitialisées et l'automate passe à l'état s'. Pour alléger la notation, une transition $t \in T_A$ sera dorénavant notée $t = (s, l, s')$ avec $l = (\delta, \gamma, m, \lambda)$. Dans la famille d'automates qu'on traite, on suppose que s' est complètement déterminé en fonction de s et l. On pourra alors noter $s' = l(s)$.

Soit A un ETA. A est supposé modéliser une spécification partielle d'un système Σ qui est en cours de spécification. On définit un ensemble de variables d'état. Un état de Σ correspond à une configuration où toute variable d'état admet une valeur connue. Soit $s \in S_A$ un état de A, s est caractérisé par un sous-ensemble des variables d'état. Les valeurs de certaines variables d'état peuvent être inconnues dans s. s représente une abstraction d'un ensemble d'états du système Σ. Lorsque A est dans l'état s, Σ peut être dans tout état dont les valeurs de ses variables d'état coïncident avec celles de s. S_A est muni d'une hiérarchie d'états définie par:

Définition 4. *Soient $s, s' \in S_A$, s' est un sous-état de s* **ssi** *l'ensemble des variables de s' est inclus dans celui de s et les variables communes entre s et s' ont les mêmes valeurs. On dit aussi que s est un super-état de s'*

Vu le lien entre les états de A et ceux de Σ, lorsque A permet un comportement à partir d'un état $s \in S_A$, alors A doit aussi permettre au-moins le même comportement à partir de tout état $s' \in S_A$ tel que s' est un sous-état de s. De manière plus formelle, par construction A doit vérifier la propriété suivante:

Propriété 1. *Soient $s_1, s_2 \in S_A$, $t = (s_1, l, s_2) \in T_A$ et $s'_1 \in S_A$. Si s'_1 est sous-état de s_1 alors il existe une transition $t' \in T_A$ telle que $t' = (s'_1, l, l(s'_1))$*

Étant donnés deux ETA A et B modélisant chacun un comportement partiel d'un même système Σ. On voudrait fusionner A et B en un seul ETA qui regroupe tous les comportements de A et de B. Pour cela, nous allons définir un nouvel opérateur noté $\oplus$:

Définition 5. *L'automate fusion $A \oplus B$ de deux ETA A et B vérifiant la propriété 1, se définit par:*

- $M_{A \oplus B} = M_A \cup M_B$,
- $H_{A \oplus B} = H_A \cup H_B$,
- $V_{A \oplus B} = V_A \cup V_B$,
- $s^o_{A \oplus B} = s^o_A \cup s^o_B$
- $S_{A \oplus B}$, $T_{A \oplus B}$ *sont respectivement initialisés par $S_A \cup S_B$ et $T_A \cup T_B$. Ensuite $S_{A \oplus B}$, $T_{A \oplus B}$ sont complétés en n'ajoutant que les états et les transitions nécessaires pour que $A \oplus B$ vérifie la propriété 1.*

Théorème 1. *L'opérateur $\oplus$ est stable, commutatif et associatif dans l'ensemble $\mathcal{E}$ des ETA vérifiant la propriété 1.*

Preuve. Étant donnés deux ETA A et B, on définit l'ETA $A \cup B$ par: $M_{A \cup B} = M_A \cup M_B, S_{A \cup B} = S_A \cup S_B, T_{A \cup B} = T_A \cup T_B, s^o_{A \cup B} = s^o_A \cup s^o_B, H_{A \cup B} = H_A \cup H_B$ et $V_{A \cup B} = V_A \cup V_B$. Étant donné un ETA A, on définit $\mathcal{C}(A)$ comme le plus petit (au sens $\subset$) ETA tel que $S_A \subset S_{\mathcal{C}(A)}$, $T_A \subset T_{\mathcal{C}(A)}$ et $\mathcal{C}(A)$ vérifie la propriété 1. $\mathcal{C}(A)$ est exactement A mais complété pour satisfaire la propriété 1. On a alors si A est un ETA vérifiant la propriété 1 alors $\mathcal{C}(A) = A$. Soient $A, B \in \mathcal{E}$, $\mathcal{C}(A \oplus B) = A \oplus B \in \mathcal{E}$ Donc $\mathcal{E}$ est stable par l'opérateur $\oplus$. De plus, $\mathcal{C}(A \oplus B) = \mathcal{C}(A \cup B)$ donc $A \oplus B = B \oplus A$ d'où $\oplus$ est commutatif dans $\mathcal{E}$. De même par l'associativité de $\cup$, on obtient l'associativité de $\oplus$ $\square$

L'application du théorème 1 permet d'assurer une intégration de scénarios qui est insensible à l'ordre d'intégration. Ceci constitue un résultat très important sur lequel se base notre approche. Ainsi, la spécification générée par intégration d'un ensemble de scénarios ne dépend pas de l'ordre de l'intégration.

4. Description de notre approche

Dans cette section, nous décrivons les différentes activités supportées par notre approche au cours du processus de la spécification du système. Notre approche permet la génération automatique d'une spécification formelle à partir des exigences des usagers. Ce processus se compose de quatre activités principales (figure 1):

1. Acquisition du domaine d'application et des scénarios
2. Intégration des scénarios
3. Analyse et vérification de cohérence
4. Simulation

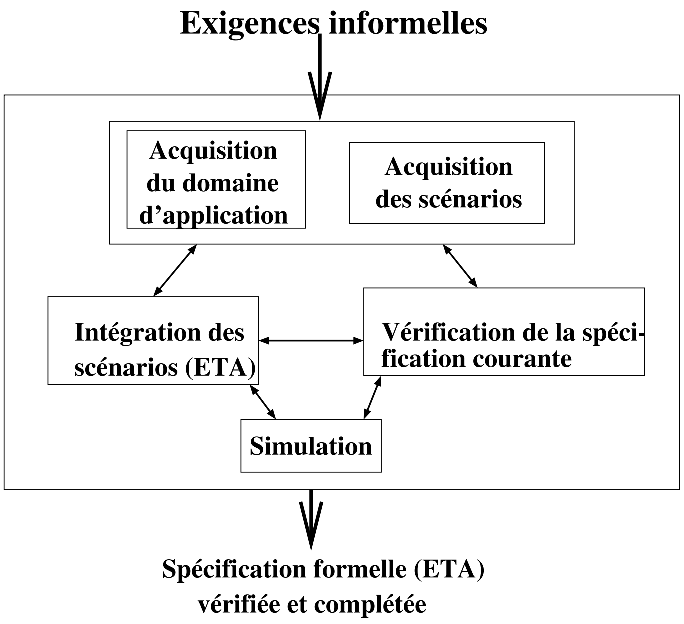


Figure 1. *Une vue générale sur le processus de spécification dans notre environnement*

4.1. *Acquisition du domaine d'application*

On entend par domaine d'application une modélisation de l'état du système à spécifier et de ses changements d'état suite aux événements observés. Dans la figure 2 on

```
IDENTIFIER_TYPE:
    terminal = {A , B , C} ;
RELATION:
    NONE() ; UP() ;
    TONE() ; DOWN();
    BUSY() ; IDLE() ;
    RING(X,Y : terminal);
    BUSYTONE(X : terminal);
    DIALING(X,Y : terminal);
    CONNECTED (X,Y : TERMINAL)
AGENT_STRUCT_Type:
    terminal : { signal = NONE | TONE | RING | BUSYTONE| DIALING ;
                 status = BUSY | IDLE ;
                 handset = DOWN | UP };
SYSTEM_VARIABLE_TYPE:
    lignes_libres = ENTIER ;
INITIALIZATION:
    SYSTEME_NAME = "Commutateur" ;
    lignes_libres := 2
```

Figure 2. *Description du domaine d'application en téléphonie : description de la caractérisation de l'état du système*

donne un exemple de description des caractéristiques de l'état du système. Cette description est formulée dans un langage que nous avons défini pour exprimer les types des données. L'exemple de la figure 2 traite le cas d'un commutateur téléphonique comportant trois postes `A`,`B` et `C`. Ces terminaux sont des agents de l'environnement du système avec lesquels il communique. L'état d'un ETA modélisant le système est composé par une concaténation des états de ces agents. L'état de chaque agent est modélisé par une structure. Dans le cas des terminaux téléphoniques, les champs de cette structure sont `signal`, `status` et `handset`. Les valeurs que peut prendre chaque champ de la structure sont des constructeurs paramétrés tels que ceux définis en CAML[1]. `lignes_libres` représente une variable globale du système, elle sera considérée comme une variable de l'ETA spécifiant le système. Notre description du domaine d'application ressemble à la description des données, des états, des opérations dans le formalisme Z [SPI 88].

L'occurrence d'un événement permet au système de changer d'état en mettant à jour sa vision sur les agents de son environnement et ses variables globales. On distingue deux types d'événements: l'émission et la réception de messages. La description d'un message se fait par l'intermédiaire d'une règle imposant des conditions minimales que l'état courant du système doit satisfaire et fournissant les modifications

1. CAML: langage fonctionnel développé par l'INRIA *http://caml.inria.fr*

à effectuer pour le changement de l'état en fonction des paramètres du message. La figure 3 montre quelques messages utilisés pour la modélisation du commutateur de la figure 2. La clause *Condition* indique les contraintes sur les variables à vérifier alors que *Update* décrit les modifications sur l'état du système. Dans le message *Pickup*, *x.handset* indique le champ *handset* de la structure d'état de l'agent *x*.

```
MESSAGE_DESCRIPTION
Pickup(x: terminal,y:entier)
  Condition:
    x.handset= DOWN
  Update:
    x.handset := UP
    x.status := Busy
Dial(x, y : terminal)
  Condition:
    x.handset = UP
    x.signal = TONE
    x.status = Busy
  Update:
    x.signal := DIALING(x,y)
```

```
SendRing(x, y : terminal)
  Condition:
    x.signal = DIALING(x,y)
    y.handset = DOWN
    y.signal = NONE
    y.status = IDLE
  Update:
    y.signal := RING(x,y)
SendEchoRing(x, y : terminal)
  Condition:
    x.handset = UP
    x.signal = DIALING(x,y)
    x.status = BUSY
  Update:
    x.signal := ECHORING(x,y)
```

Figure 3. *Description du domaine d'application en téléphonie: description des messages*

4.2. *Acquisition des scénarios*

La description du domaine permet la spécification des données et les scénarios fournissent la description des processus. Le comportement du système est décrit de manière incrémentale sous forme d'un ensemble de scénarios. Un scénario est une description partielle du comportement, il est constitué de deux parties:

– les préconditions qui forment un ensemble de conditions sur l'état du système et représentent les conditions minimales sur l'état courant du système avant l'exécution du scénario.

– une liste d'interactions dont chaque élément comprend un couple *stimulation/réaction*, des contraintes temporelles et des contraintes sur les variables, et une réaction du système en cas d'expiration des contraintes temporelles. La stimulation est représentée par la réception d'un message alors que la réaction est constituée par l'envoi d'un ou de plusieurs messages. La forme générale d'une interaction est décrite dans la figure 4 sous forme d'un MSC. Le temporisateur T1 indique le délai au plus tôt pour accepter de recevoir le message `Stimulation`. Cependant, cette réception n'est plus acceptée après le délai au plus tard indiqué par T2. A l'expiration de T2, le système envoie le message `Réaction d'expiration` pour éviter un blocage. Indiquer les délais au plus tôt et au plus tard dans la description d'une interaction ainsi que la

Réaction d'expiration est optionnel. Les messages Stimulation, Réaction et Réaction d'expiration sont des instanciations partielles ou totales de messages décrits dans la définition du domaine d'application. L'instanciation totale d'un message consiste à donner des valeurs constantes à tous ses paramètres. Cette instanciation est partielle si le message comporte au moins un paramètre variable. Ce paramètre est considéré comme une variable locale de l'interaction. La clause contraintes sur variables à la figure 4 représente une conjonction de contraintes respectant la définition 3 section 3 avec ν soit une variable globale du système, soit une variable locale de l'interaction, soit une variable de l'état d'un agent.

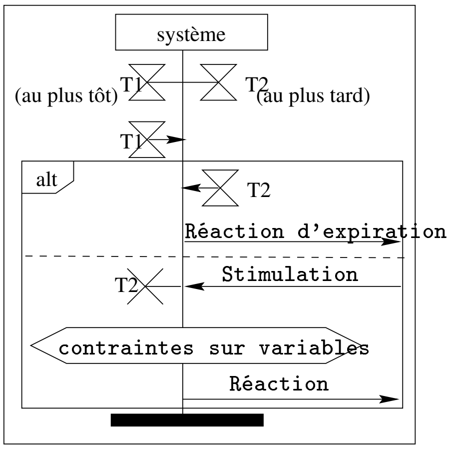


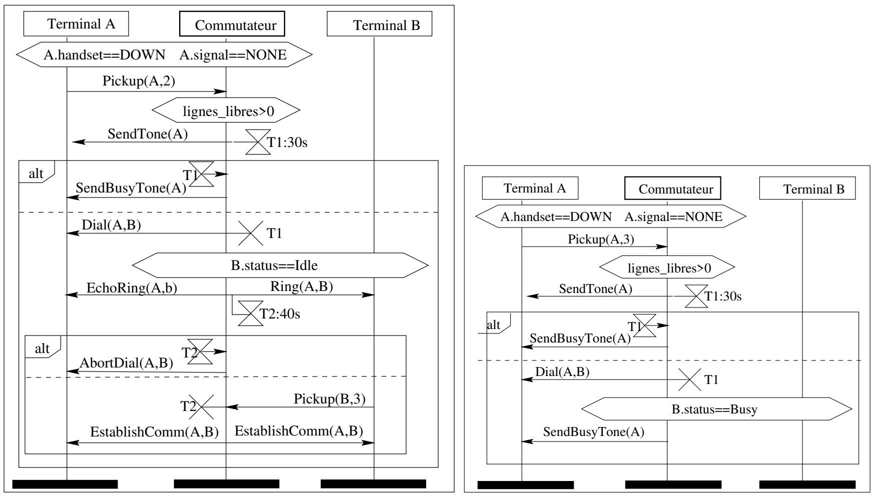
Figure 4. *un MSC illustrant le schéma général d'une interaction*

L'extraction des scénarios se fait à partir des exigences informelles et leur représentation utilise une extension par les données du formalisme des MSC (Z.120) [ENG 99]. Les MSC permettent l'abstraction des traitements tout en mettant en évidence les aspects de la communication et les conditions qui les régissent dans un contexte particulier. L'un des inconvénients des MSC est le manque de caractérisation formelle de l'état du système. Dans notre cas, ce manque est complété par la description du domaine. Par contre, l'une des difficultés majeures reste l'extraction des "bons" scénarios. La figure 5 présente deux scénarios où le terminal A appelle terminal B mais dans (a) B est libre alors que dans (b) il est occupé.

4.3. *Intégration des scénarios*

Cette activité est la plus importante dans le processus de spécification, car elle permet de regrouper les comportements partiels du système acquis sous forme de scénarios et d'en synthétiser une spécification globale qui reproduit les scénarios intégrés.

La spécification obtenue peut contenir de nouveaux scénarios qui résultent des chevauchements des comportements décrits dans les scénarios intégrés. Ces nouveaux scénarios peuvent être détectés et s'ils sont indésirables alors ils pourront être interdits. Cependant l'algorithme d'intégration peut être configuré pour que la spécification générée contienne exactement les scénarios fournis.



(a) appelé libre (b) appelé occupé

Figure 5. *Scénarios d'établissement d'un appel*

La spécification du système est exprimée sous forme d'ETA. Le choix de ce modèle se justifie par le fait qu'il permet de représenter convenablement les deux aspects véhiculés dans les scénarios, à savoir le contrôle temporisé et le flux de données à travers les échanges des messages contraints dans le temps et par les valeurs des variables et paramètres.

L'activité d'intégration des scénarios opère en deux phases. La première transforme un scénario en un ETA qui vérifie la propriété 1. Cet ETA est équivalent à son scénario. L'ETA d'un scénario a pour ensemble d'états, les états par lesquels passe le système durant l'exécution du scénario et, pour ensemble de transitions, les événements du scénarios. L'ETA du scénario généré peut être utilisé dans la phase 2 en le fusionnant avec l'ETA de la spécification courante en utilisant l'opérateur $\oplus$ (section 3).

4.3.1. *Génération de l'automate d'un scénario*

Cette opération a pour but de vérifier la cohérence du scénario en isolation avant de le traduire en un ETA. Ceci se déroule en deux phases.

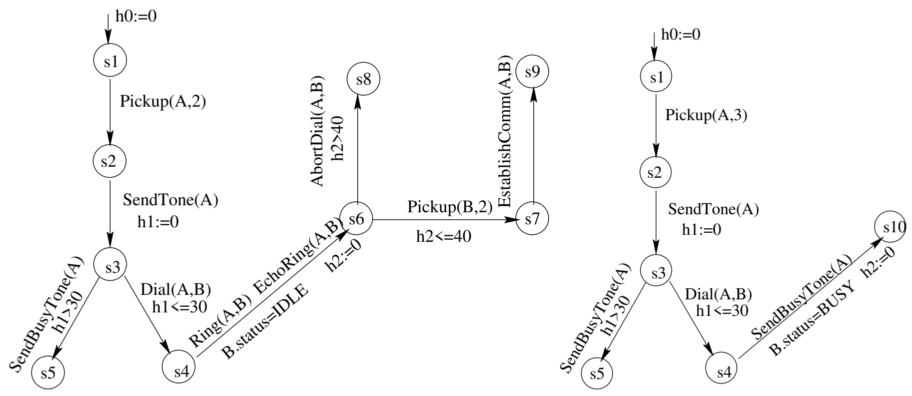
Phase 1 la vérification intra-scénario qui permet de s'assurer de l'exécutabilité séquentielle des interactions du scénario. Si une interaction ne peut pas s'exécuter dans l'état d'arrivée de l'interaction qui la précède, un message indique la cause qui l'empêche et la phase échoue. L'état initial du scénario peut faire abstraction des valeurs de certaines variables d'état qui le constituent. Les valeurs de ces variables sont supposées inconnues ou "Don't care". Lorsqu'une interaction impose une contrainte sur les variables d'état, cette contrainte est propagée en amont jusqu'à l'état initial du scénario. Ceci peut avoir comme effet l'instanciation de certaines variables qui étaient inconnues.

Phase 2 après le succès de la phase 1, le scénario est traduit en un ETA par la fonction *scenario2ETA*. L'algorithme de traduction consiste à insérer dans un automate vide toutes les interactions du scénario.

1. **fonction** *scenario2ETA*(Sc: scenario)
2. $\quad A$:= ETA vide
3. $\quad$($precondition$, $[I_1, \ldots, I_n]$):= Sc
4. $\quad precondition$:= s_{input}
5. $\quad$**Pour chaque** $I \in [I_1, \ldots, I_n]$ **faire**
6. $\qquad$($t_{input}, t_{output}, t_{expire}$):=determinerInteractionTrans(s_{input}, I)
7. $\qquad$**Pour chaque** $\alpha \in \{input, output, expire\}$ **faire**
8. $\qquad\quad (s_\alpha, l_\alpha, s'_\alpha) := t_\alpha$
9. $\qquad\quad S_A := S_A \cup \{s_\alpha, s'_\alpha\}$;$T_A := T_A \cup \{t_\alpha\}$
10. $\qquad\quad$mettre à jour A pour que la propriété 1 soit vérifiée
11. $\qquad s_{input} := s'_{output}$
12. $\quad$**retourner** A

La fonction *determinerInteractionTrans(*s_{input}, I*)* à la ligne 6 permet de synthétiser trois transitions représentant l'interaction I et à partir de l'état s_{input}. En se référant au schéma général d'une interaction à la figure 4, t_{input}, t_{output} et t_{expire} représentent respectivement la réception du message `Stimulation`, l'envoi du message `Réaction` et l'envoi du message `Réaction d'expiration` respectivement. s_{input} est l'état de départ de la transition t_{input}. Son état d'arrivée s'_{input} est déterminé en appliquant le message `Stimulation` à l'état s_{input}. De même s'_{expire} est obtenu en appliquant le message `Réaction d'expiration` sur l'état s_{expire} qui est égal à s_{input}. On peut aussi facilement remarquer que les états s_{output} et s'_{input} sont égaux. En effet, le comportement décrit par l'interaction indique que t_{output} est franchissable après t_{input}. De plus, les contraintes temporelle de t_{input} traduisent les délais au plus tôt et au plus tard de l'interaction. Par contre, les contraintes temporelles de t_{expire} sont obtenues par le complément de celles de t_{input}. Ceci permet d'éviter le blocage du système si la transition t_{input} ne peut plus être franchie. t_{output} a comme garde, les contraintes sur variables de l'interaction. Ceci permet entre autres de vérifier si le paramètre reçu dans t_{input} satisfait une condition avant de franchir t_{output}.

L'application de cet algorithme aux scénarios de la figure 5 permet de générer leur ETA respectifs (figure 6).



(a) appelé libre (b) appelé occupé

Figure 6. *ETA généré à partir des scénarios de la figure 5.*

4.3.2. *Fusion de deux spécifications*

Cette opération s'applique sur deux ETA en utilisant l'opérateur de fusion $\oplus$. Souvent l'un de ces deux ETA constitue la spécification en cours et l'autre c'est l'ETA d'un scénario qu'on veut ajouter. Étant donnés deux ETA, A et B, vérifiant la propriété 1, l'algorithme construisant l'ETA fusion $A \oplus B$ se déroule en deux phases: l'une permet l'insertion des transitions en entrée et l'autre celles de sortie. La stratégie suivie consiste à étendre T_A par T_B de la façon suivante:

```
fonction fusion(A, B : ETA)
    A ⊕ B := A
    Pour chaque transition d'entrée t ∈ T_B faire
        ajouterTransEntree(t, A ⊕ B, B)
    Pour chaque transition de sortie faire
        ajouterTransSortie(t, A ⊕ B)
    retourner A ⊕ B
```

La fonction *ajouterTransEntree(t,A,B)* insère la transition $t = (s, l, s') \in T_B$ dans l'ETA A. Ceci commence par l'insertion des états s et s' dans S_A tout en effectuant les mises à jour nécessaires pour préserver la propriété 1. Ensuite, On cherche dans T_A une transition $t' = (s, l', s')$ compatible avec t. On dit que deux transitions t et t' sont compatibles si elles ont les mêmes contraintes temporelles et les mêmes contraintes sur les variables et si leurs messages sont unifiables en les paramétrant pour que les deux transitions deviennent égales. En cas de paramétrisation, certaines contraintes sur les paramètres ajoutés sont insérées dans les transitions suivant t dans A et celles suivant t' dans B. Ceci dans le but de préserver les comportements des ETA A et B après la paramétrisation. Si aucune transition de T_A n'est compatible avec t, celleci est ajoutée à T_A. Ensuite, On effectue les changements nécessaires pour garder la propriété 1 valide sur A.

Par contre, la fonction *ajouterTransSortie(*t, A*)* permet de faire le même traitement que la fonction *ajouterTransEntree* sauf que dans ce cas on exige que les transitions de sortie t et t' soient identiques Ceci est dû au fait que le système contrôle ses transitions de sortie.

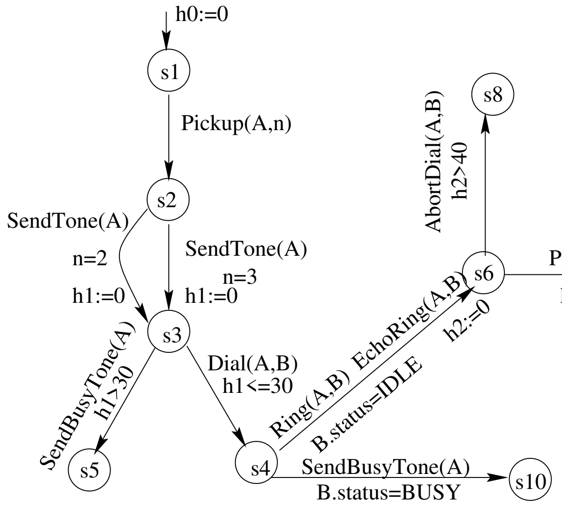


Figure 7. *ETA obtenu par la fusion des automates de la figure 6.*

On donne à la figure 7 un ETA modélisant un commutateur téléphonique. Cette spécification est obtenue par fusion des ETA de la figure 6. On remarque facilement que cet ETA contient les comportements décrits dans les deux scénarios de la figure 5.

4.4. *Analyse et vérification de la cohérence*

Les scénarios aussi bien que la description du domaine peuvent contenir des contradictions ou des oublis qui peuvent se propager jusqu'aux spécifications générées. L'activité d'analyse et de vérification a pour rôle de détecter ces incohérences et de retracer leurs causes afin de faciliter leurs corrections. Cette activité est réalisée en trois étapes:

4.4.1. *Vérification intra-scénario*

Elle vise à découvrir toutes les incohérences liées à un scénario en tant qu'entité isolée avant son insertion dans la spécification courante. On commence par s'assurer que le scénario constitue une séquence exécutable. Ceci est effectué pendant la traduction du scénario en un ETA. Un scénario est non exécutable si l'une de ses interactions ne l'est pas à cause de certaines préconditions des messages ou de contraintes qui ne sont pas satisfaites. Cette situation peut être corrigée de deux façons: par une modification des préconditions du message qui posent le problème ou par une modification du scénario. Cette dernière peut modifier les contraintes insatisfaites ou insérer une interaction qui rend le scénario exécutable. Une analyse d'accessibilité de l'ETA du scénario permet ensuite de détecter certaines incohérences liées au contraintes temporelles et aux contraintes sur les variables du système.

4.4.2. *Vérification inter-scénario*

Cela consiste à vérifier que l'ETA de la spécification courante est cohérente. La spécification courante peut regrouper plusieurs scénarios. Cette phase a pour but de s'assurer que les scénarios sont cohérents les uns par rapport aux autres. Cette vérification concerne, pour le moment, les aspects suivants :

Cohérence temporelle et cohérence des contraintes sur les variables: Certains états de l'ETA sont inaccessibles à cause des contraintes temporelles ou des contraintes sur les variables qui sont insatisfaisables. De telles contraintes sont détectées par une analyse d'accessibilité basée sur les automates des régions [ALU 94] et sur les techniques de propagation des contraintes. Mais, à cause du problème de l'exécutablité des traces, certains états non accessibles risquent ne pas être détectés. Lorsqu'une incohérence est détectée, on retrace les scénarios qui en sont la cause afin de les corriger. Pour faciliter le retraçage d'un scénario à partir de l'ETA de la spécification, chaque transition comporte l'identificateur du scénario qui la contenait. Plusieurs outils de vérification des systèmes temps-réel tels que Kronos [YOV 97] ou UPPAAL [BEN 96] peuvent éventuellement être utilisés pour la vérification de l'aspect temporel des ETA générés.

Résolution du non déterminisme: Il s'agit du non-déterminisme au sens du test [FUJ 91]. Il peut être causé de deux manières : soit à cause d'une erreur soit à cause de l'abstraction. Le premier cas est rencontré lorsqu'il y a une interaction entre les fonctionnalités (feature interaction). Il peut aussi être causé par un conflit lors du chevauchement de deux scénarios. Par contre, le non déterminisme causé par l'abstraction peut être éliminé lors du raffinement de la spécification. Pour détecter ce type de non déterminisme, on parcourt l'ETA de la spécification pour chercher les états desquels plusieurs transitions de sortie partent. On vérifie si les contraintes temporelles et les contraintes sur les variables permettent plus d'une transition de sortie. Dans ce cas on émet un message à l'analyste qui va décider soit de protéger ou de raffiner ce non déterminisme dans le cas d'une abstraction, soit de corriger l'erreur dans le cas d'interactions indésirables.

4.4.3. *Aide à la complétion*

Cette étape vise à éviter certaines erreurs de type "*réception non spécifiée*" en complétant la spécification générée. On parle de réception non spécifiée lorsque le système dans un état donné reçoit un message qu'il ne peut pas traiter, car il n'a aucune transition étiquetée par la réception de ce message à partir de cet état. Cette situation mène à un blocage. Souvent, c'est durant cette étape que des oublis dans la spécification sont détectés surtout ceux qui concernent le traitement des erreurs de comportement de l'environnement ou des messages à ignorer.

4.5. *Simulation*

A tout moment du processus de création de services, la spécification courante peut être simulée. Ceci revient à exécuter l'ETA de la spécification à partir d'un état du système. Les transitions de cet ETA sont interprétées comme un ensemble de règles gérées par un moteur d'inférence pour éliminer les conflits lorsque plusieurs transitions sont franchissables à partir d'un même état du système.

5. Conclusion

Nous avons proposé un environnement de création de services basé sur l'approche scénario. Il permet la compréhension des exigences et la génération automatique, à partir des scénarios d'utilisation, d'un ETA comme spécification formelle exécutable du système. Les scénarios intégrés véhiculent un comportement partiel enrichi par des contraintes temporelles et des contraintes sur les variables et les paramètres des messages échangés. Comme seconde contribution, notre environnement offre aussi la vérification, la complétion et la simulation de la spécification générée, il forme ainsi un outil CASE pour l'ingénierie des exigences. Les algorithmes d'intégration décrits dans cet article ont été implantés dans un prototype avec Objective CAML. D'autres travaux sur la vérification sont en cours de réalisation.

Nous comptons maintenant expérimenter notre approche sur des cas réels et dans d'autres domaines d'application. D'abord les algorithmes d'intégration peuvent être améliorés pour réduire l'effet de l'explosion d'états, par l'intermédiaire d'un pliage des ETA générés. La propriété 1 sera toujours vérifiée par les ETA générés afin de satisfaire les hypothèses de notre fondement théorique. Ensuite, notre algorithme d'intégration peut être adapté pour offrir une intégration explicite en imposant un opérateur d'intégration entre certains scénarios. On distingue déjà deux opérateurs l'un pour l'intégration séquentielle et l'autre pour l'intégration d'alternative. Ce type d'intégration explicite est utile pour la description de l'existant ou lorsqu'on veut générer des spécifications qui comportent exactement les scénarios intégrés et dans un ordre prédéterminé. Enfin, nous allons développer un moteur d'inférence pour l'activité de simulation. Étant donnés un ETA et un état du système, ce moteur d'inférence aura pour but de déterminer la prochaine action que le système va exécuter.

Remerciements
Nous tenons à remercier Marc Feeley pour les discussions constructives et son support lors de la réalisation du projet.

6. Bibliographie

[ALU 94] ALUR R., DILL D., « A Theory of Timed Automata », *Theoretical Computer Science*, vol. 126, 1994, p. 183-235.

[BEN 96] BENGTSSON J., LARSEN K. G., LARSSON F., PETTERSSON P., « UPPAAL: a tool suite for automatic verification of real-time systems », *Lecture Notes in Computer Science*, vol. 1066, 1996, page 232.

[BIE 76] BIERMANN A. W., KRISHNASWAMY R., « Constructing programs from example computations », *IEEE Trans. Software Engineering*, vol. SE-2, n° 9, 1976, p. 141–153.

[CAR 92] CARROLL J. M., ROSSON M. B., « Getting Around the Task-Artifact Cycle: How to Make Claims and Design by Scenario », *ACM Transactions on Information Systems*, vol. 10, n° 2, 1992, p. 181–212.

[DAN 97] DANO B., « Une démarche d'ingénierie des besoins orientée objet guidée par les cas d'utilisation », PhD thesis, Université de Nantes, Novembre 1997.

[ENG 99] ENGELS A., FEIJS L., MAUW S., « MSC and Data: dynamic variables », *SDL ROFUM 99*, 1999.

[FUJ 91] FUJIWARA S., BOCHMANN G., KHENDEK F., AMALOU M., GHEDAMSI A., « Test Selection Based on Finite-State Models », *IEEE Transactions Software Engineering*, vol. SE-17, NO. 6, 1991, p. 591-603.

[GLI 95] GLINZ M., « An Integrated Formal Model of Scenarios Based on Statecharts », *Software Engineering - ESEC'95. Proceedings of the 5th European Software Engineering Conference*, Springler LNCS 989, 1995, p. 254–271.

[HAR 87] HAREL D., « STATECHARTS: A Visual Formalism for Complex Systems », *Science of Computer Programming*, vol. 8, 1987, p. 231-274, Elsevier Science Publishers.

[HSI 94] HSIA P., SAMUEL J., GAO J., KUNG D., TOYOSHIMA Y., CHEN C., « Formal Approach to Scenario Analysis », *IEEE Software*, vol. 11, 1994, p. 33–41.

[KHR 98] KHRISS I., ELKOUTBI M., KELLER R. K., « Automating the Synthesis of UML Statechart Diagrams from Multiple Collaboration Diagrams », BEZIVIN J., EDITORS P. A. M., Eds., *UML'98: Beyond the Notation*, 98.

[KOS 94] KOSKIMIES K., MÄKINEN E., « Automatic Synthesis of State Machines from Trace Diagrams », *Software-Practice and Experience*, vol. 24, n° 7, 1994, p. 643-658.

[NAR 92] NARDI B. A., « The use of Scenarios in Design », *SIGCHI Bulletin*, vol. 24, n° 4, 1992.

[RAT 97] RATIONAL-SOFTWARE-CORPORATION, MICROSOFT, HEWLETT-PACKARD, ORACLE, STERLING, MCI, UNISYS, ICON, INTELLICORP, I-LOGIXRATIONAL SOFTWARE CORPORATION SANTA CLARA C., UML notation guide, version 1.1, 1997.

[SOM 95] SOMÉ S., DSSOULI R., VAUCHER J., « From Scenarios to Timed Automata: Building Specifications from Users Requirements », *Proceedings of the 2nd Asia Pacific Software Engineering Conference (APSEC'95)*, IEEE, 1995.

[SOM 97] SOMÉ S. S., « Dérivation de Spécification à partir de Scénarios d'Interaction », PhD thesis, Université de Montréal, Juin 1997.

[SPI 88] SPIVEY J. M., *Understanding Z: A Specification Language and Its Formal Semantics*, Cambridge Tracts in Theoret, Cambrige University Press, UK, 1988.

[YOV 97] YOVINE. S., « Kronos: A verification tool for real-time systems », *In Springer International Journal of Software Tools for Technology Transfer*, vol. 1, n° 1/2, 1997.