Université de Montréal

Génération automatique d'une spécification formelle à partir de scénarios temps-réels

par
Aziz Salah

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Thèse présentée à la Faculté des études supérieures
en vue de l'obtention du grade de
Philosophiæ Doctor (Ph.D.)
en informatique

Avril 2002

Université de Montréal
Faculté des études supérieures

Cette thèse intitulée :

Génération automatique d'une spécification formelle
à partir de scénarios temps-réels

présentée par :

Aziz Salah

a été évaluée par un jury composé des personnes suivantes :

Houari Sahraoui	(président-rapporteur)
Rachida Dssouli	(directeur de recherche)
Guy Lapalme	(co-directeur de recherche)
Ferhat Khendek	(membre du jury)
Loïc Hélouët	(examinateur externe)
Samuel Pierre	(représentant du doyen de la FES)

Thèse acceptée le 31 mai 2002

Sommaire

Les systèmes réactifs temps-réels forment une catégorie de systèmes qui interagissent avec leur environnement selon des contraintes temporelles strictes. La spécification formelle de tels systèmes est l'activité qui consiste à produire un modèle formel décrivant leurs comportements. Devant la difficulté de cette activité, les développeurs passent souvent directement de la spécification informelle à l'implémentation, réduisant ainsi possiblement la fiabilité des systèmes.

Cette thèse vise à développer des méthodes pour synthétiser de manière automatique une spécification formelle pour des systèmes réactifs temps-réels en se basant sur l'approche scénario. Celle-ci décrit le comportement du système par un ensemble de scénarios qui seront intégrés pour synthétiser une spécification globale. Chaque scénario décrit le comportement du système dans une situation particulière et représente une spécification partielle du système. L'approche scénario représente ainsi une stratégie de diviser pour régner qui facilite l'activité de spécification car nous n'en considérons qu'une partie à la fois. La spécification synthétisée est exprimée sous forme d'un automate temporisé sécuritaire.

Nous distinguons deux aspects, l'un statique et l'autre dynamique, dans la modélisation d'un système réactif temps-réel. L'aspect statique représente la modélisation des caractéristiques du système en utilisant des événements, des variables discrètes et des horloges à valeurs utilisant des nombres réels. Les variables définissent les propriétés non temporelles du système. Les horloges permettent de chronométrer le temps écoulé depuis un événement donné et servent de référence pour permettre ou interdire l'exécution des actions du système. La modélisation de l'aspect dynamique représente la spécification de comportement. Nous représentons l'unité élémentaire du comportement comme étant une *action-règle* qui décrit comment l'état du système change suite à l'exécution d'une action. Nous définissons la sémantique d'un ensemble d'actions-règles par un

système de transitions étiquetées auquel nous associons un automate temporisé sécuritaire. Afin de réduire l'explosion combinatoire de cet automate nous le minimisons en utilisant une relation de bisimulation.

Les scénarios de comportement du système sont décrits selon un nouveau modèle convivial. Chaque scénario représente un ensemble de traces de l'exécution du système. Ce modèle permet de cacher l'aspect formel de la description du comportement tout en tenant compte de sa dépendance par rapport au temps. La sémantique d'un scénario est reliée à celle d'un ensemble d'actions-règles qui représente sa forme canonique. Nous établissons ainsi la relation entre notre modèle de scénario et les automates temporisés sécuritaires.

L'intégration des scénarios consiste à les fusionner en un seul automate temporisé sécuritaire. Nous proposons deux catégories de méthodes d'intégration des scénarios : implicite et explicite. Pour chacune nous définissons la relation de conformité entre les scénarios d'origine et l'automate temporisé sécuritaire résultant de leur intégration. Les méthodes d'intégration implicite fusionnent les scénarios sans aucune contrainte supplémentaire sur leur ordonnancement. La méthode intégration explicite est basée sur la spécification des directives d'intégration explicite qui sont considérées comme des contraintes sur l'ordonnancement des scénarios. Nous proposons un algorithme pour la satisfaction des directives d'intégration. Les scénarios qui résultent de cette étape sont intégrés en utilisant les méthodes d'intégration implicite. Notre approche permet la combinaison des méthodes d'intégration implicite et explicite dans la même spécification.

Notre approche supporte aussi plusieurs tâches de vérification au niveau syntaxique, au niveau du comportement des scénarios ainsi qu'au niveau de l'automate temporisé sécuritaire résultant. En cas de détection d'erreurs nous pouvons toujours retracer le ou les scénarios qui en sont la cause.

Mots-clés : intégration de scénarios, ingénierie des exigences, spécification formelle, système temps-réel, automate temporisé.

Summary

A real-time reactive system interacts with its environment under strict timing constraints. The formal specification of such systems is a difficult task that often designers prefer to avoid to move directly from informal specifications to implementation. The detection of design errors and thus the reliability of the system become difficult to evaluate thus increasing the development costs.

This thesis develops automatic methods for synthesis of formal specification in the case of real-time reactive systems. We use a scenario based approach which describes the system behavior by scenarios that are integrated into a global specification. Scenarios describe a system behavior in a particular situation and represent partial specifications of the system. Scenarios based approaches facilitate the task of specification as they use a divide and conquer strategy. Our work focus on development of methods of integration of scenarios which allow automatic synthesis of a formal specification for a real-time reactive system. This formal specification is expressed in the form of a secure timed automaton.

A real-time reactive system conceptualization encompasses static and dynamic aspects. Static aspect deals with system properties with the use of discrete variables, clocks and events. Discrete variables conceptualize the time independent properties of a system. Clocks measure time and are used as time references to enable or disable the execution of an action. Dynamic aspect describes the behavior of the system and uses the static aspect as description language. We call a *rule-action* the elementary unit of the description of a system behavior. A rule-action states how the state of a system changes after the execution of an action. We define a global semantics of a rule action set by a labeled transition system to which we associate a secure timed automaton. We reduce the combinatorial explosion of the size of this automaton using a bisimulation relation based minimization algorithm.

The description of scenarios is based on a new suite model which hides the formal aspect of the description of a system behavior. However, this scenario model supports the description of a

timed constrained behavior. A scenario is a set of execution traces and its semantics is defined by a set of rule-actions which represents the canonical form of this scenario. We thus establish the relation between scenarios and secure timed automata.

Integration of scenarios merges them into a single secure timed automaton. We propose two types of integration methods : implicit integration and explicit integration methods. For each, we define the conformance relation between the initial scenarios and the resulting timed automaton. Implicit integration merge scenarios without any explicit scheduling constraint. Explicit integration uses explicit integration directives which are constraints on the execution order of scenarios. We propose an algorithm to solve the problem of satisfaction of the explicit integration directives. Scenarios which result from this phase, are merged into a timed automaton using implicit integration methods. Furthermore, our approach supports the combination of implicit and explicit integration methods in the same specification. Our scenario based approach supports syntax verification, scenario verification and result of integration verification. We can trace back to the faulty scenario if an error is detected.

Keywords : integration of scenarios, requirement engineering, formal specification, real-time system, timed automata.

Table des matières

Table des figures

Liste des tableaux

Liste des acronymes

ATS ..... Automate temporisé sécuritaire

ATS-EAR ..... Automate temporisé sécuritaire d'un ensemble d'actions-règles

FSM..... Machine à états finis

GIE ..... Graphe d'Intégration Explicite

IOA ..... Automate à entrées et sorties

STE ..... Système de transitions étiquetées

STE-EAR ..... Système de transitions étiquetées d'un ensemble d'actions-règles

MSC ..... Message sequence chart

PSED ..... Problème de la Satisfaction d'un Ensemble de Directives

UML ..... Unified Modeling Language

Dédicace

À la mémoire de ma mère
À mon père
À ma douce épouse
À Hmizat

Remerciements

Je remercie le bon Dieu de m'avoir aidé à mener à terme ce travail. Je tiens aussi à remercier tous ceux qui ont participé et m'ont soutenu tout au long de ce travail.

Je désire exprimer ma profonde gratitude à Rachida, ma directrice de recherche, cette femme de grande modestie, que je remercie pour sa confiance, son soutient moral et ses suggestions éclairées qui ont été d'une grande utilité dans l'avancement de ce travail. Je tiens également à remercier Guy, mon codirecteur de recherche pour sa patience, ses remarques pertinentes ainsi que ses conseils d'expert.

Je voudrais aussi remercier les autres membres du jury MM Loïc Hélouët, Houari Sahraoui, Ferhat Khendek et Samuel Pierre pour avoir accepté de juger ce travail et pour l'intérêt qu'ils lui ont accordé.

Je tiens de même à remercier :

- France Télécom qui a assuré le financement de ce projet de recherche.
- le Doyen de la Faculté des sciences et techniques de Tanger Monsieur Mustapha Bennouna, ainsi que tous les collègues, pour m'a voir faciliter de venir à l'Université de Montréal pour faire cet thèse.
- Yassir et Dhafer pour leur aide dans le développement de l'outil SCE_NA .
- et mes compagnons de labeur Jawad, Salem, Abdslam, Abdel, ... pour les discussions fructueuses, leurs aides, les moments agréables ...

Mes vifs remerciements sont adressés à mes parents et à ma douce épouse pour leur générosité inépuisable et leur patience. Très reconnaissant, je tiens à demander pardon à mon fils Hamza et à sa maman pour le sacrifice que je leur ai imposé.

Chapitre 1

Introduction

1.1 Motivations

L'invention de l'ordinateur et de l'Internet sont parmi les révolutions technologiques qui ont profondément marqué le $20^{ième}$ siècle. Ainsi fut la naissance d'une nouvelle technologie que nous désignons actuellement par *système informatique* qui envahit tous les domaines de la vie moderne. Un système informatique combine des composantes matérielles et logicielles qui assurent des activités aussi variées que le contrôle de procédés industriels, la gestion du trafic aérien ou urbain, etc .... Leur impact socio-économique est devenu très fort dans la mesure où il occupe des places de plus en plus stratégiques et critiques aux seins des organisations et du quotidien des individus. Une classe importante des systèmes informatiques est représentée par les *systèmes réactifs temps-réels*, systèmes en interaction continue avec leur environnement. Leur caractérisation de "temps-réel" indique que ces systèmes doivent agir sous des délais et des contraintes temporelles strictes. Dans certains cas, la violation d'une contrainte temporelle peut avoir des conséquences catastrophiques.

En pratique le développement des systèmes réactifs temps-réels passe de la spécification informelle à l'implémentation à cause de l'absence d'outils de spécification formelle. De plus l'application d'une méthode formelle est difficile d'autant que la taille des systèmes est grande. Par conséquent, la fiabilité du système devient difficile à évaluer. Pour certains systèmes une telle incertitude sur la fiabilité du comportement est non tolérée ; ainsi l'unique alternative est la spécification formelle.

Nous désignons par "*spécification formelle*" l'utilisation d'une technique de modélisation mathématique pour exprimer le comportement d'un système informatique. Nous énumérons quelques avantages de la spécification formelle :

- Elle est basée sur un modèle mathématique bien établi, ainsi elle permet d'éviter l'ambiguïté du langage naturel ou d'une représentation informelle.
- Elle permet la diversité de conception et l'exploration de plusieurs alternatives avant de s'engager dans l'implémentation.
- Elle est utilisée dans plusieurs étapes du cycle de vie du système informatique telles que :
  - la vérification : consiste à prouver de manière mathématique que la spécification du système est correcte. L'accomplissement de cette tâche nécessite la disponibilité d'une spécification formelle.
  - l'implémentation : la spécification formelle peut servir pour la génération automatique de programmes.
  - le test de conformité : la spécification formelle représente une référence pour l'implémentation. Le test de conformité consiste à vérifier si l'implémentation est conforme à sa spécification ou non.
  - la simulation : si la spécification du système est exprimée dans un modèle exécutable, dans ce cas la spécification du système représente un prototype qui permet la simulation de son comportement.

Le but de cette thèse est la génération *automatique* d'une spécification formelle dans le cas des systèmes réactifs temps-réels.

La réalisation de ce but pourrait se faire en envisageant des techniques de traitement automatique du langage naturel pour formaliser une spécification informelle. Malheureusement ces techniques ne sont pas encore prêtes à répondre à ce besoin. Une autre alternative consiste à extraire un ensemble de spécifications formelles partielles à partir des spécifications informelles. Ces spécifications partielles sont ensuite intégrées en une spécification formelle globale. Nous réduisons ainsi la complexité de la tâche de spécification formelle car nous ne considérons qu'une partie à la fois. Ces spécifications partielles représentent des *scénarios* du comportement du système en conception.

1.2 Objectifs et contributions

Dans cette thèse, nous développerons des méthodes d'intégration de scénarios de comportement pour générer de manière automatique, pour un système réactif temps réel, une spécification formelle exprimée sous forme d'un automate temporisé sécuritaire [NSY92].

Un scénario est une description partielle du comportement d'un système dans une situation donnée. Il représente une partie de la spécification de ce système. Un exemple de scénario dans le cas d'un commutateur téléphonique peut être informellement décrit par le texte de la figure 1.1

> Lorsque le poste A est non occupé, si le commutateur reçoit le message $pickup(A)$ alors il envoie la tonalité au poste A. Si le poste A appelle B avant 30 unités de temps alors le commutateur envoie la sonnerie lorsque le poste B est libre. Mais si le poste A ne fait rien pendant 30 unités de temps alors le commutateur lui envoie le signal occupé. Lorsque le terminal du poste B est en train de sonner, si celui-ci décroche avant 60 unités de temps, le commutateur établit la communication sinon il l'annule et envoie le signal occupé.

FIG. 1.1 – Description en langue naturelle d'un scénario d'établissement d'une communication décrivant le comportement d'un commutateur téléphonique simplifié

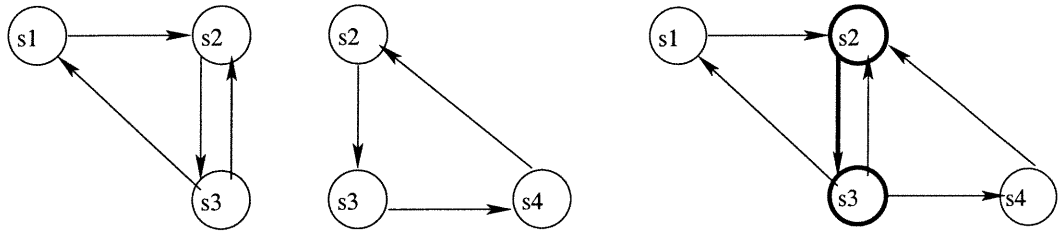
L'intégration de scénarios construit une spécification formelle et globale qui regroupe tous les comportements de ces scénarios.

Par conséquent, il faudrait avoir un moyen qui permette de caractériser une action dans un scénario dans le but de la reconnaître dans d'autres scénarios. Cette reconnaissance d'éléments commun entre deux scénarios représente le ciment qui rassemble "les morceaux" de la spécification globale. Certaines approches [HSG+94, Gli95, EK98], utilisent pour cet effet un étiquetage manuel. Les figures 1.2(a) et 1.2(b) illustrent deux scénarios représentés sous forme de systèmes de transitions. Une possibilité de l'intégration de ces deux scénarios est donnée à la figure 1.2(c).

Pour obtenir la spécification représentée à la figure 1.2(c), nous nous sommes basés sur :

- l'étiquetage des états
- le fait que cet étiquetage est suffisant pour caractériser les éléments qui sont communs à plusieurs scénarios.

De manière générale, nous classifions les méthodes d'intégration de scénarios en deux catégories. La première est celle des méthodes d'intégration implicites. Elles sont basées sur la ca-



(a) (b) (c)

FIG. 1.2 – Les systèmes de transitions (a) et (b) modélisent deux scénarios du comportement d'un système. Le système de transitions (c) représente une possibilité d'intégration des scénarios (a) et (b). Les éléments tracés avec un trait continu fort sont communs aux deux scénarios.

ractérisation des éléments des scénarios pour construire une spécification globale [KM94, Som97, DSVS99, EK98, SDL01a]. La seconde catégorie est celle des méthodes d'intégration explicites qui peuvent aussi être appelées déclaratives car dans ce cas, le concepteur fournit les scénarios et comment les rassembler en utilisant par exemple des schémas ou des opérateurs d'intégration qui décrivent l'ordonnancement de l'exécution des scénarios [HSG+94, Gli95, Dan97, SDL01b]. Cependant, à notre connaissance, aucun travail sur les méthodes d'intégration des scénarios en un système de transitions n'a défini de manière formelle la relation de conformité entre les scénarios et la spécification globale résultante (de l'intégration) où la sémantique des scénarios sera garanties.

Nous visons une intégration *automatique* des scénarios. Par conséquent, il faut absolument avoir en entrée un modèle formel pour la représentation des scénarios. Ainsi quelles sont les exigences en terme d'expressivité sur le modèle de représentation de nos scénarios ? Les scénarios représentent une spécification partielle du comportement d'un système réactif *temps-réel* donc une représentation explicite des contraintes temporelles dans les scénarios est nécessaire. Le modèle des automates temporisés sécuritaires [HNSY94] n'est pas nouveau mais nous allons définir une sémantique opérationnelle formelle pour ce modèle d'automates qui sera directement liée à celle de notre nouveau modèle des scénarios.

Les scénarios sont décrits en utilisant des variables qui concrétisent les propriétés d'un système

réactif temps-réel. Nous allons utiliser deux types de variables : des horloges comme celles utilisées dans les automates temporisés [AD94, HNSY94] et des variables discrètes pour modéliser les propriétés non temporelles du comportement. Ainsi nous proposons un langage pour la description des scénarios.

Nous allons définir de manière formelle deux familles de méthodes d'intégration de scénarios, l'une est implicite et l'autre explicite. Ces deux méthodes peuvent être combinées dans la même spécification où certains scénarios seront intégrés par le mode implicite alors d'autres le seront par le mode explicite. Notre approche formelle nous a aussi permis la détection de certaines incohérences et contradictions entre les scénarios. Ceci est possible grâce à une représentation interne des scénarios sous une forme appelée *action-règles* [SDL01a] et qui est à la base de tous les modes d'intégration de scénarios dans notre approche.

Dans certains cas, la spécification générée par l'intégration des scénarios peut souffrir d'une explosion combinatoire dans la taille de son automate temporisé sécuritaire. Nous allons proposer un algorithme pour la réduction de la taille de cet automate.

Nos méthodes d'intégration des scénarios et l'algorithme de réduction de la spécification résultante ont été implantés dans un outil baptisé S_CE_NA pour l'aide à la spécification formelle des systèmes réactifs temps-réel.

1.3 Organisation du document

Nous avons résumé les motivations et les objectifs du travail, ainsi que les solutions que nous proposons. Nous présentons ici l'organisation du document.

Nous consacrons le **chapitre 2** à l'introduction des systèmes à transitions. Nous traitons dans l'ordre les modèles des réseaux de Pétri, des systèmes de transitions étiquetées et des Statecharts. Ces modèles nous seront utiles pour l'étude de l'état de l'art des travaux sur la génération de spécification par intégration des scénarios de comportement. Dans le cas des systèmes de transitions étiquetées, nous traitons aussi les relations de conformité permettant d'établir des équivalences entre deux spécifications.

Dans le **chapitre 3** nous définissons la syntaxe et la sémantique de deux variétés d'auto-

mates temporisés. Premièrement les automates temporisés de Büchi [AD94] dont la sémantique est définie par un système de transition muni d'un ensemble d'états d'acceptation. Ensuite, nous présentons les automates temporisés sécuritaires ainsi que leur sémantique opérationnelle.

Dans le **chapitre 4** nous présentons une revue de la littérature sur les travaux d'intégration des scénarios en une spécification. Pour chaque travail exposé, nous allons suivre une méthodologie qui consiste à étudier l'acquisition et la représentation des scénarios, leur intégration, ainsi les tâches de vérification et d'analyse des scénarios si elles sont couvertes. De même nous aborderons aussi l'utilisation des méthodes formelles dans ces travaux. Ensuite nous présentons une synthèse comparative entre les différentes approches présentées.

Le **chapitre 5** décrit le fondement théorique de notre approche. Nous commençons par présenter une nouvelle modélisation des systèmes réactifs temps-réel qui est composée de deux descriptions : l'une est statique et l'autre est dynamique. La description statique représente la description du domaine d'application, elle fournit les éléments du langage pour décrire le comportement d'un système qui forme sa description dynamique et qui est exprimée par un ensemble d'actions-règles. La définition d'une sémantique opérationnelle pour les actions-règles nous permet de construire un automate temporisé sécuritaire formellement équivalent. Nous traitons ensuite le réduction de cet automate temporisé.

Dans le **chapitre 6**, nous présentons un nouveau modèle de scénarios pertinent pour la représentation des interactions d'un système réactif temps-réel avec son environnement. Ensuite, nous définissons la forme canonique d'un scénario par un ensemble d'actions-règles qui exprime sa sémantique formelle. Par l'intermédiaire de la forme canonique d'un scénario nous lui associons un automate temporisé sécuritaire qui modélise son comportement.

Le **chapitre 7** présente les modes d'intégration implicite pour lesquels aucune contrainte sur l'ordonnancement des scénarios intégrés n'est explicitement spécifiée. Nous allons d'abord présenter deux modes d'intégration implicite qui sont purs ensuite nous allons montrer comment ces deux modes peuvent être combinés dans la même spécification. Pour chaque mode d'intégration implicite nous définissons la relation de conformité qui lui est propre et qui garantit son exactitude.

Dans le **chapitre 8**, nous définissons le mode d'intégration explicite en spécifiant des directives d'intégration qui représentent des contraintes d'ordonnancement sur l'exécution de certains scénarios. Nous allons proposer un algorithme permettant de déterminer de manière automatique les contraintes nécessaires et suffisantes pour sur-contraindre les scénarios afin de satisfaire les directives d'intégration explicite. Les scénarios résultant seront alors intégrés en utilisant les modes d'intégration implicite.

Le **chapitre 9** présente l'outil *SCENA* que nous avons développé pour l'aide à la spécification par intégration des scénarios. L'outil implante les méthodes d'intégration implicite et explicite des scénarios du chapitre 7 et 8 ainsi que l'algorithme de réduction des spécifications résultantes de l'intégration des actions-règles présenté dans le chapitre 5. L'outil *SCENA* permet aussi certaines tâches de vérification.

Finalement, au **chapitre 10**, nous résumons les principales contributions de cette thèse et nous présentons quelques directions possibles pour les recherches futures.

Chapitre 2

Les Systèmes de Transitions : Modèles de Spécification Formelle

Dans ce chapitre nous allons introduire les systèmes de transitions et donner quelques définitions qui nous seront utiles par la suite. Basés sur des modèles mathématiques, les systèmes de transitions jouissent d'une sémantique formelle. Ils servent ainsi souvent de modèle pour exprimer des spécifications formelles du comportement en général. Plus particulièrement, ils permettent de représenter avec un niveau d'abstraction satisfaisant les systèmes réactifs. Le fait d'avoir une sémantique formelle est nécessaire pour tout traitement automatique de la spécification. Nous déduisons ainsi notre intérêt pour les systèmes de transitions.

Les modèles des systèmes de transitions sont nombreux. Nous nous restreignons dans ce chapitre à décrire certains systèmes que nous qualifions de "classiques" car il ne permettent pas la modélisation du temps. Nous allons présenter, dans l'ordre, les réseaux de Pétri, les systèmes de transitions étiquetées et les Statecharts qui sont sont représentatifs de cette famille de systèmes de transitions "classiques" et qui sont utilisés dans les approches scénario.

2.1 Les réseaux de Pétri

Les réseaux de Pétri[1] [GR85] constituent un formalisme mathématique particulièrement adapté à la modélisation des systèmes où les aspects d'événements concurrents et d'évolutions simul-

[1]Cette appellation est en hommage au père fondateur Carl Adam Petri qui les a introduits dans sa thèse "*Kommunikation mit Automaten*" en 1962 à l'Université de Darmstadt

tanées sont présents. Ils sont utilisés dans la spécification et la validation des protocoles de communication en particulier et des systèmes distribués en général. Il permettent l'évaluation des performances des systèmes discrets et même la conception des interfaces homme-machine. Plusieurs raisons ont favorisé l'utilisation des réseaux de Pétri. D'une part, c'est un formalisme qui a atteint une certaine maturité vu le nombre d'outils disponibles pour l'édition graphique et la validation des modélisations exprimées en réseaux de Pétri. D'autre part, leur représentation graphique les rend faciles à comprendre et à utiliser. De plus les réseaux de Pétri permettent la vérification de certaines propriétés des systèmes qu'il décrivent. Par exemple, ils vérifient des propriétés de sûreté, de vivacité ou de non-interblocage.

Définition 2.1. *Un réseau de Pétri est un quadruplet $R = (P, T, Pre, Post)$, où :*

- *P est un ensemble fini de places,*
- *T est un ensemble fini de transitions,*
- *$Pre : P \times T \longrightarrow \mathbb{N}$ est la fonction incidence avant,*
- *$Post : P \times T \longrightarrow \mathbb{N}$ est la fonction incidence arrière sachant que*

Définition 2.2. *Un réseau de Pétri marqué est le couple $N = (R, M)$, où :*

- *R est un réseau de Pétri,*
- *$M : P \longrightarrow \mathbb{N}$ est une application qui associe à chaque place $p \in P$ un certain nombre de marques, appelées* jetons. *$M(p)$ est le marquage de la place p et M_o est l'application de marquage initial.*

Le marquage peut être ainsi représenté par un vecteur ayant comme dimension le nombre de places.

Un réseau de Pétri est un graphe orienté où les places sont les noeuds du graphe et où les transitions sont représentées par des traits horizontaux reliés aux places par des arcs (figure 2.1). Les places jouent le rôle de variables d'état et les transitions, contrôlées par les événements, permettent aux jetons de changer de places. La fonction Pre associe au couple $(p, t) \in P \times T$ un nombre de jetons nécessaires pour franchir la transition t à partir de la place p. La valeur de $Pre(p, t)$ est par défaut égale à 1 si elle n'est pas indiquée sur le graphe.

Définition 2.3. *Une transition t est dite* franchissable (enabled) *ssi :*

$$\forall p \in P : M(p) \geq Pre(p, t)$$

Une fois la transition franchie, le terme $Post(p, t)$ représente le nombre de jetons créés dans les places d'arrivée de t. Par exemple, pour franchir la transition "Préparer le café" de la figure

2.1, il faut que les deux places en amont comportent chacune au moins un jeton. Après que cette transition soit franchie, un jeton est retranché de chaque place en amont puis ajouté à chaque place en aval.

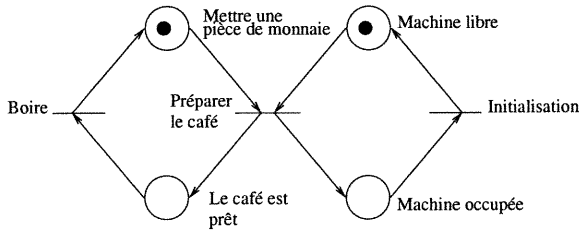


FIG. 2.1 – Un réseau de Pétri modélisant une machine à café

Cependant, les réseaux de Pétri présentent quelques limitations :

- Le temps n'est représenté que de manière causale ce qui revient à une représentation qualitative. Une représentation quantitative permet par exemple de spécifier des durées min/max pour tirer une transition.
- Plusieurs problèmes concernant la vérification des propriété (vivacité, blocage) dans les réseaux de Pétri restent encore ouverts dans le cas général [Fur93] bien qu'ils soient décidables pour certaines familles de réseaux de Pétri particulières.

Afin de palier à certaines de ces limitations et d'augmenter leur expressivité, plusieurs extensions des réseaux de Pétri ont été proposées. Parmi les plus répandues nous citons :

Les réseaux de Pétri colorés [Jen92] : Les réseaux de Pétri colorés donnent plus de sémantique aux jetons en leur associant des couleurs différents représentant des valeurs de données arbitraires. Une transition devient franchissable non seulement selon la disponibilité des jetons dans les places situées à son entrée mais selon leurs couleurs aussi. Cette extension permet d'aboutir à des description plus concises et d'éviter beaucoup de duplications typiques dans le modèle de base.

Les réseaux de Pétri avec transitions à prédicat [Fur93] : C'est une grande variété des réseaux de Pétri dans laquelle des pré-conditions et des post-conditions sont imposées pour franchir une transition. Les pré-conditions se manifestent comme des prédicats de la logique du premier ordre alors que les post-conditions sont exprimées sous forme d'affectations de variables

Les réseaux de Pétri objet [LK91, LKP93] : Dans cette extension, les jetons sont des objets associés à des classes de style orienté objet qui sont définies par des types de données et des méthodes. Les places sont caractérisées par les objets qu'elles ont accueillis selon un ordre implicite de leurs arrivées. Les transitions exécutent les méthodes de ces objets.

Les réseaux de Pétri temporisés et stochastiques Les réseaux de Pétri peuvent simuler des processus temporisés en ajoutant des contraintes de délai soit dans les places [CR83], soit dans les transitions [GMMP91]. De manière conventionnelle, l'une de ces deux alternatives est choisie puisque l'une peut être exprimée en fonction de l'autre. Si les délais sont distribués d'une manière aléatoire alors il s'agit des réseaux de Pétri stochastiques [MBC+95]. Par contre si les délais sont distribués de manière déterministe, on parle dans ce cas des réseaux de Pétri temporisés [Hil89, Chr84].

2.2 Systèmes de transitions étiquetées

Les systèmes de transitions étiquetées sont utilisés dans plusieurs travaux théoriques tels que le test [Bri88]. Ils ont aussi servi pour décrire la sémantique d'un certain nombre de langages de spécification comme CCS [Mil89], CSP [Hoa85] ou LOTOS [ISO88].

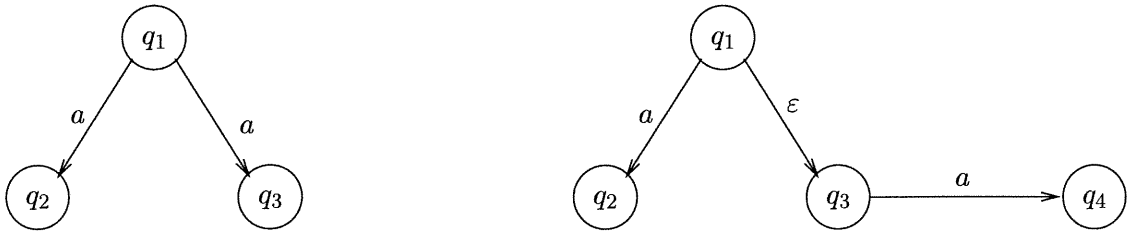
2.2.1 Définition du modèle

Définition 2.4. *Un systèmes de transitions étiquetées est un triplet* $\Sigma = (Q, Q_o, \rightarrow)$ *où :*

- Q *est un ensemble non vide d'états du système de transitions étiquetées* Σ
- $Q_o \subset Q$ *est un ensemble non vide des états initiaux de* Σ
- $\rightarrow \subset Q \times E \cup \{\varepsilon\} \times Q$ *est la relation de transition de* Σ*, sachant que* E *est l'ensemble d'événements (étiquettes) observables et que* ε *représente un événement invisible (interne).*

Une transition $(q, a, q') \in \rightarrow$ *est notée* $q \xrightarrow{a} q'$.

Nous utilisons parfois l'abréviation STE pour désigner un système de transitions étiquetées. Dans la théorie des automates, le futur, c'est la transition suivante qui peut être choisie à partir d'un état [Pha94]. Selon cette théorie, un STE est dit non déterministe lorsqu'il a plusieurs transitions étiquetées par le même événement et à partir du même état (figure 2.2.a). Par contre, dans les théories d'algèbre de processus, le futur, représente la prochaine transition à événement observable. Dans ce cas un STE est dit non déterministe lorsqu'il est non déterministe au sens de la théorie des automates mais aussi lorsqu'il comporte le schéma de la figure 2.2.b.



(a) Non déterminisme au sens de la théorie des automates

(b) Non déterminisme au sens des théories d'algèbre de processus

FIG. 2.2 – Non déterminisme dans les STEs

2.2.2 Sous modèles des systèmes de transitions étiquetées

Le modèle des STEs, est un modèle général qui englobe plusieurs sous-modèles plus spécifiques nous ne citons que les automates à entrée et sortie et les machines à états finis. Le premier modèle, communément connu sous son appellation anglaise "Input Output Autoamata" (IOA) [LT88], est un STE dont l'ensemble d'événements est composé de deux catégories d'événements : les entrées et les sorties. Certains auteurs ajoutent une troisième catégorie, les événements internes. Cette discrimination des événements donne une sémantique particulière aux IOAs permettant la modélisation de composantes formant un système concurrent asynchrone [LT88, Dri00].

Le deuxième modèle, celui des machines à états finis (FSM), a été utilisé pour le développement des circuits séquentiels synchrones [Gil62, Koh78] et dans les protocoles de communication et plus spécifiquement dans le test de ces protocoles. Les transitions dans ce modèle sont étiquetées par deux événements couplés : l'un représente l'événement *stimulus* (entrée) et l'autre la *réaction* (sortie).

2.2.3 Traces dans les systèmes de transitions étiquetées

La notion de traces est très importante dans les systèmes de transitions car elle représente une exécution du système.

Définition 2.5. *Soit* $\Sigma = (Q, Q_o, \rightarrow)$ *un LTS.* $\sigma = a_1 a_2 \dots a_n$ *est une trace à partir d'un état* q *de* Σ *ssi il existe* $q_0, q_1, \dots q_n$ *des états de* Σ *tels que* $q_i \xrightarrow{a_i} q_{i+1}$ *pour* $0 \leq i \leq n$ *et* $q = q_0$. σ *est la*

trace d'un chemin dans Σ entre q_0 et q_n. Nous notons $TR(q)$ l'ensemble des traces à partir de q.

Un système qui est modélisé par un STE , est considéré dans un environnement. Les événements internes du système ne sont pas observables pour l'environnement. Une trace observable est une trace où tous les événements internes sont supprimés. Nous notons $OTR(q)$ l'ensemble des traces observables à partir de l'état q.

2.2.4 Composition parallèle

La communication entre deux processus qui sont représentés sous forme de STEs est définie par leur composition parallèle.

Définition 2.6. *Soient $\Sigma_i = (Q_i, Q_{oi}, \rightarrow_i)$ un STE et E_i son ensemble d'événements observables pour $i \in \{1, 2\}$. La composition parallèle de Σ_1 et Σ_2 est un STE noté $\Sigma_1 || \Sigma_2 = (Q, Q_o, \rightarrow)$ et défini par :*

- $Q = Q_1 \times Q_2$
- $Q_o = Q_{o1} \times Q_{o2}$
- *La relation de transition $\rightarrow$ de $\Sigma_1 || \Sigma_2$ est définie par les règles suivantes :*

$$\forall a \in (E_1 \backslash E_2) \cup \{\varepsilon\} \; . \; q_1 \xrightarrow{a}_1 q_1' \;\; implique \;\; \forall q_2 \in Q_2, (q_1, q_2) \xrightarrow{a} (q_1', q_2) \qquad (2.1)$$

$$\forall a \in (E_2 \backslash E_1) \cup \{\varepsilon\} \; . \; q_2 \xrightarrow{a}_2 q_2' \;\; implique \;\; \forall q_1 \in Q_1, (q_1, q_2) \xrightarrow{a} (q_1, q_2') \qquad (2.2)$$

$$\forall a \in (E_1 \cap E_2) \; . \; (q_1 \xrightarrow{a}_1 q_1' \;\; et \;\; q_2 \xrightarrow{a}_2 q_2') \;\; implique \;\; (q_1, q_2) \xrightarrow{a} (q_1', q_2') \qquad (2.3)$$

La communication dans ce modèle est synchrone puisque les STEs communicants doivent se synchroniser sur leurs événements communs. Ce type de communication est dit synchronisation par rendez-vous. Il est utilisé dans CCS [Mil89], CSP [Hoa85] et LOTOS [ISO88].

2.2.5 Relations de conformité dans les systèmes de transitions étiquetées

Ces relations de conformité permettent de comparer deux STEs entre eux afin d'établir une équivalence. Plusieurs relations de conformité ont été proposées dans littérature [dH84, Hen85, Par81, Arn92]. Nous rappelons ici les définitions des quatres relations les plus utilisées : la bisimulation [Par81], la simulation [Arn92], l'équivalence observationnelle [Mil80] et l'équivalence de traces [Hoa85, BHR84].

Bisimulation

Définition 2.7. *Soient $\Sigma_i = (Q_i, Q_{oi}, \rightarrow_i)$ un LTS pour $i \in \{1,2\}$. Une bisimulation entre Σ_1 et Σ_2 est une relation binaire $R \subset Q_1 \times Q2$ entre Σ_1 et Σ_2 telle que :*

(i a) $\forall q_1 \in Q_1, \exists q_2 \in Q_2.(q_1, q_2) \in R$

(i b) $\forall q_2 \in Q_2, \exists q_1 \in Q_1.(q_1, q_2) \in R$

(ii a) $\forall q_1 \in Q_1 \,.\, q_1 \xrightarrow{a}_1 q_1'$ *et* $(q_1, q_2) \in R$ *implique* $\exists q_2' \in Q_2 \,.\, q_2 \xrightarrow{a}_2 q_2'$ *et* $(q_1', q_2') \in R$

(ii b) $\forall q_2 \in Q_2 \,.\, q_2 \xrightarrow{a}_2 q_2'$ *et* $(q_1, q_2) \in R$ *implique* $\exists q_1' \in Q_1 \,.\, q_1 \xrightarrow{a}_1 q_1'$ *et* $(q_1', q_2') \in R$

$(q_1, q_2) \in R$ peut aussi être noté $q_1 R q_2$.

Simulation

Une relation binaire R qui ne satisfait que les points *(i a)* et *(ii a)* de la définition 2.7, est une simulation entre Σ_1 et Σ_2. Dans ce cas, Σ_2 permet au moins tous les comportements de Σ_1. Nous disons que Σ_2 simule Σ_1.

Équivalence observationnelle

Appelée aussi bisimulation faible, l'équivalence observationnelle a été introduite par Milner [Mil80]. Elle est définie comme une bisimulation mais qui ne tient pas compte des événements internes. Elle représente ainsi une relation moins fine que la bisimulation et plus fine que l'équivalence de traces.

Définition 2.8. *Soient $\Sigma_i = (Q_i, Q_{oi}, \rightarrow_i)$ un STE pour $i \in \{1,2\}$. Une relation d'équivalence observationnelle entre Σ_1 et Σ_2 est une relation binaire $R \subset Q_1 \times Q2$ entre Σ_1 et Σ_2 telle que :*

(i a) $\forall q_1 \in Q_1, \exists q_2 \in Q_2.(q_1, q_2) \in R$

(i b) $\forall q_2 \in Q_2, \exists q_1 \in Q_1.(q_1, q_2) \in R$

(ii a) Pour tout q_1 appartenant à Q_1 et pour tout q_2 appartenant Q_2 tels que (q_1, q_2) appartient à R, si $\varepsilon^n a \varepsilon^m$ est la trace d'un chemin dans Σ_1 entre q_1 et q_1' alors il existe q_2' appartenant à Q_2 tel que (q_1', q_2') appartient à R et il existe un chemin dans Σ_2 entre q_2 et q_2' dont la trace est $\varepsilon^{n'} a \varepsilon^{m'}$ avec $n, m, n', m' \geq 0$.

(ii b) Pour tout q_2 appartenant à Q_2 et pour tout q_1 appartenant Q_1 tels que (q_1, q_2) appartient à R, si $\varepsilon^n a \varepsilon^m$ est la trace d'un chemin dans Σ_2 entre q_2 et q_2' alors il existe q_1' appartenant à Q_1 tel que (q_1', q_2') appartient à R et il existe un chemin dans Σ_1 entre q_1 et q_1' dont la trace est $\varepsilon^{n'} a \varepsilon^{m'}$ avec $n, m, n', m' \geq 0$.

Équivalence de trace

Cette équivalence correspond au concept d'équivalence de langages dans la théorie des automates [Gar89].

Définition 2.9. *Soient* $\Sigma_i = (Q_i, Q_{oi}, \rightarrow_i)$ *un STE pour* $i \in \{1, 2\}$. Σ_1 *et* Σ_2 *sont équivalents pour la trace ssi toute trace observable dans* Σ_1 *est une trace observable dans* Σ_2 *et réciproquement. C'est dire, ssi :*

$$\bigcup_{q_1 \in Q_1} OTR(q_1) = \bigcup_{q_2 \in Q_2} OTR(q_2)$$

L'équivalence de trace est la relation de conformité minimale qui peut relier deux STEs.

2.3 Les statecharts

Les Statecharts [Har87] représentent une extension du formalisme conventionnel des machines à états finis (FSM). Ils se basent sur un formalisme graphique qui permet la visualisation d'une description à différents niveaux d'abstraction et de détails ; ce qui peut faciliter la gestion et la compréhension des spécifications ayant un très grand nombre d'états.
Le développement du formalisme des Statecharts vise à remédier à certaines limitations des machines à états en traitant la concurrence, la communication et en introduisant la notion d'une hiérarchie d'abstraction. Ceci a permis de réduire l'explosion du nombre des états, lors de la représentation graphique, dont souffrent les machines à états.

D'une manière technique, chaque état du Statechart peut être récursivement décomposé en :

- soit un autre Statechart selon une décomposition hiérarchique,
- soit deux ou plusieurs Statecharts parallèles reliés par un opérateur **AND** et considérés comme des *threads*[2].
- Au niveau le plus bas les Statecharts sont représentés par des FSMs.

La communication entre les Statecharts se fait par diffusion globale des événements. Si la réception d'un événement par un Statechart à un état donné déclenche une transition, alors celle-ci sera exécutée et pourra générer un autre événement qui sera à son tour diffusé à tous les Statecharts actifs. Les événements ne sont ni stockés dans des files ni sauvegardés et ceux qui n'ont déclenché aucune transition sont ignorés et perdus.

[2]Harel [Har87] appelle cela l'*orthogonalité*

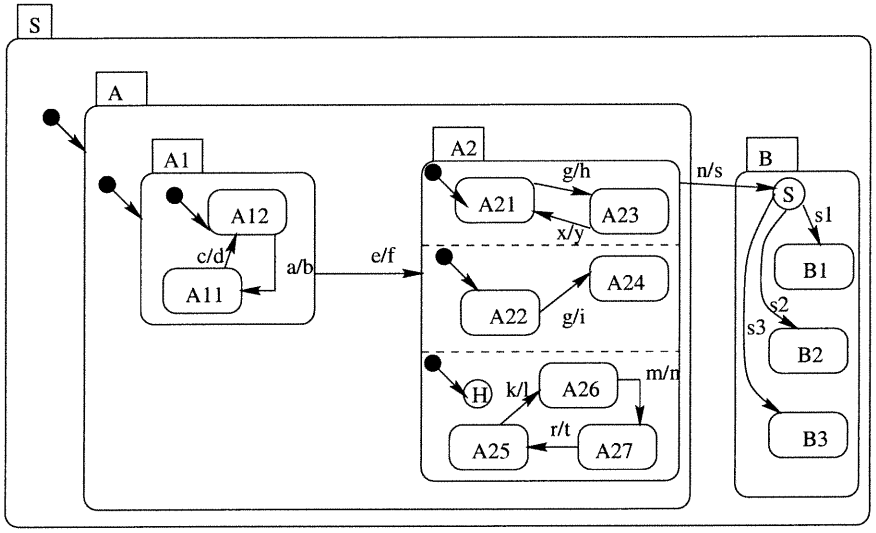
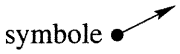


FIG. 2.3 – Un exemple des Statecharts illustrant leur syntaxe graphique

La séquence d'événements (a,c,e,g) *produit la séquence d'actions suivante :* (b,d,f,i et h). A2 *est un état AND (parallèle), c'est pourquoi l'événement* g *donne lieu à deux actions* i *et* h.

La forme générale d'une transition s'écrit sous la forme $\mu = \alpha(P)/\beta$ *i.e* lorsque l'événement α s'est réalisé et si la condition P est satisfaite alors l'action β est exécutée. En terme de la théorie des automates, α appartient à l'alphabet des entrées et β à celles des sorties. La condition P empêche la transition μ d'être franchie lorsque cette première est fausse.

Les Statecharts comportent d'autres mécanismes qui renforcent leur expressivité et leur concision :

- le symbole  représente la transition initiale qui montre l'état initial du Statechart,
- le mécanisme **H** (troisième thread de A2 dans la figure 2.3) permet de sauvegarder l'historique d'un Statechart. Ceci permet à un Statechart de poursuivre son exécution à partir de là où il l'a laissée lors de sa dernière visite,
- le mécanisme **C** (pour *condition*) permet de déterminer un état de début d'exécution à l'entrée d'un satecharts selon une condition fournie,
- le mécanisme **S** (pour *sélection*) sert lorsque plusieurs états sont regroupés dans un Statechart en permettant de choisir un état spécifique selon l'alternative précisée dans la transition tel qu'indiqué dans l'état B de la figure 2.3. Le sens donné au choix ici est celui du *ou exclusif* puisque le système doit être exclusivement dans l'un des états du Statechart en question.

Par ailleurs la sémantique des Statecharts est définie dans [HN96]. Cette sémantique a été utilisée dans l'outil STATEMATE [HLN+88].

2.4 Conclusion

Dans ce chapitre nous avons passé en revue différents modèles de systèmes de transitions, donné leur sémantique et montré les catégories de systèmes qu'ils peuvent décrire. Nous consacrons le chapitre suivant aux modèles des automates temporisés puisque le modèle cible dans notre approche d'intégration des scénarios est un automate temporisé.

Chapitre 3

La modélisation du Temps et les Automates Temporisés

Le chapitre précédent a présenté des modèles "classiques" où le temps est modélisé de manière qualitative. Ces modèles spécifient un ordre causal entre les événements mais ne permettent pas la spécification d'une durée séparant deux événements. Le besoin de représenter le temps de manière quantitative s'est fait sentir avec le développement des systèmes temps-réels. Le comportement de tels systèmes ne dépend pas seulement des événements, mais aussi des délais et des instants où ces événements ont lieu. Ainsi le comportement d'un système temps-réel respecte des contraintes temporelles quantitatives strictes.

Dans un premier temps, la section 3.1 introduit les différentes représentations du temps dans les systèmes à transitions. Ensuite nous présentons à la section 3.2 le modèle de base des variétés d'automates temporisés et leur syntaxe commune. Les sections 3.3 et 3.4 décrivent respectivement les automates temporisés de Büchi et les automates temporisés sécuritaires.

3.1 Modèles de représentation du temps

Dill [Dil90] a regroupé les modèles de la représentation du temps dans les systèmes à transitions en trois catégories : le modèle de *temps discret* [EMSS92], le modèle à *horloge fictive* globale [Ost93] et le modèle de *temps dense* [AD94].

– Le modèle de temps discret considère que le temps est une séquence monotone et croissante

de nombres entiers naturels. Chaque processus exécute au plus une seule action entre deux instants entiers. Pour s'accommoder à cette contrainte, il suffit d'adapter la durée qui correspond à l'intervalle de temps séparant deux entiers successifs pour fixer la bonne granularité.

- Le modèle à horloge fictive est similaire au modèle précédent sauf que le temps est supposé continu même si on ne tient compte que de la partie entière des instants des actions. La partie entière se réfère à une horloge fictive globale et discrète. Cette approximation donne lieu à un modèle imprécis du temps.
- Le modèle de temps dense représente les instants des événements par des nombres réels. Le temps dans ce modèle progresse de façon continue, monotone et non bornée.

Les modèles de temps discret et à horloge fictive s'adaptent aisément aux formalismes des système à transitions. Par contre le temps dans les processus physiques n'est pas discret mais plutôt continu. En exprimant le temps sous forme d'un nombre d'une *unité de temps indivisible* fixée à priori, les modèles de temps discret et à horloge fictive s'avèrent inexacts car ils se basent sur une approximation entière des instants. Cette approximation représente l'inconvénient majeur de ces modèles.

Le modèle de temps dense permet de préserver l'aspect continu du temps en représentant les instants sous forme de nombres réels. Ce modèle est plus difficile à adapter au formalisme des systèmes de transitions car le nombre d'instants et donc le nombre d'états sont indénombrables. Alur et Dill [AD90] ont défini un modèle à représentation finie permettant l'implantation d'un modèle dense du temps. Le modèle défini se base sur une extension des systèmes de transitions étiquetées en lui ajoutant un ensemble de variables réelles qui mesurent l'écoulement du temps.

3.2 Graphe temporisé

La structure du graphe temporisé est commune à tous les modèles d'automates temporisés [AD90, NSY92, ACD93, AD94, Alu99]. Un graphe temporisé permet une représentation finie des automates temporisés. De manière informelle un graphe temporisé admet la structure d'un STE étendu par un ensemble de variables réelles appelées *horloges*. Les transitions du graphe temporisé sont étiquetées par un triplet composé d'un événement, d'une contrainte d'horloges et d'une assignation d'horloges. Dans la suite de cette section, nous allons définir ce triplet ainsi que la syntaxe de ses éléments afin de donner la définition d'un graphe temporisé.

3.2.1 Horloges et valuation d'horloge

Les horloges sont des variables dont les valeurs croissent uniformément avec le temps.Étant donné un ensemble d'horloges H, une valuation[1] d'horloges θ est une application de H vers $\mathbb{R}^+$. $\theta(h)$ représente la valeur associée par la valuation d'horloges θ à l'horloge h. Toute valuation d'horloges θ est aussi interprétée comme un vecteur de $\mathbb{R}^{|H|}$ dans le but d'une représentation géométrique.

Pour tout réel positif d , $\theta + d$ est la valuation d'horloges définie par $(\theta + d)(h) = \theta(h) + d$ pour tout $h \in H$. $\theta + d$ permet d'avancer toutes les horloges d'un délai d. Nous notons $\Theta(H)$ l'ensemble des valuations d'horloges de H.

3.2.2 Contrainte d'horloge

Définition 3.1. *Les contraintes d'horloges sont les formules respectant la grammaire suivantes :*

$$\varphi ::= h\#c \mid h - h'\#c \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \neg\varphi \mid True \mid False \tag{3.1}$$

avec $\# \in \{\leq, <, =, \geq, >\}$*, où h,h' sont des horloges et c est un entier naturel appartenantà* $\mathbb{N}$*.* $\Phi(H)$ *désigne l'ensemble des contraintes sur les horloges de* H*.*

Définition 3.2. *Les contraintes positives d'horloges sont les formules respectant la grammaire suivantes :*

$$\varphi ::= h\#c \mid h - h'\#c \mid \varphi \wedge \varphi \mid True \mid False \tag{3.2}$$

avec $\# \in \{\leq, <, =, \geq, >\}$*, où h,h' sont des horloges et c est un entier appartenantà* $\mathbb{N}$*.* $\Phi^+(H)$ *désigne l'ensemble des contraintes positives sur les horloges de* H*.*

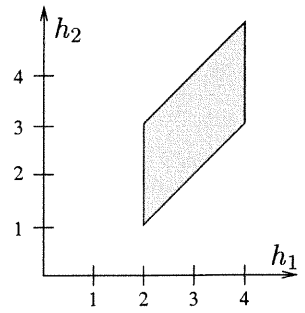
Une contrainte positive d'horloges peut toujours s'écrire comme une conjonction de terme $h\#c$ et $h - h'\#c$. Toute contrainte d'horloges peut s'écrire comme une disjonction de contraintes positives d'horloges.

Étant donnés $\theta \in \Theta(H)$ et $\varphi \in \Phi(H)$, $\varphi(\theta)$ représente la valeur de vérité de la contrainte φ évaluée en θ. θ satisfait φ si et seulement si $\varphi(\theta)$ a la valeur vrai. De même, $[\varphi]$ représente la zone des valuations d'horloges qui satisfont la contrainte φ. $[\varphi]$ est en fait la zone géométrique

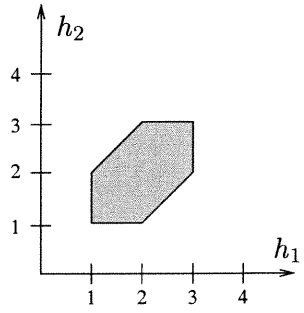
[1] appelé aussi interprétation d'horloges par certains auteurs

dont φ est l'équation caractéristique. La figure 3.1 illustre la représentation des zones de certaines contraintes pour $H = \{h_1, h_2\}$.

Exemple 3.1. *Considérons la contrainte* $\varphi = h_1 - h_2 < 3 \wedge h_2 < 2$. $\varphi(0.5, 1)$ *est vraie par contre* $\varphi(4.5, 1)$ *est fausse*



Représentation de la zone
$[h_1 \geq 2 \wedge h_1 \leq 4 \wedge h_1 - h2_{\leq} 1 \wedge h_1 - h2_{\geq} - 1]$



Représentation de la zone
$[h_1 \geq 1 \wedge h_1 \leq 3 \wedge h_1 - h2_{\leq} 1 \wedge$
$h_1 - h2_{\geq} - 1 \wedge h_2 \geq 1 \wedge h_2 \leq 3]$

FIG. 3.1 – Illustration de la zone associée à une contrainte

3.2.3 Assignation d'horloge

Nous considérons λ une sous-partie de H. λ représente l'assignation d'horloges permettant la remise à zéro des horloges appartenant à λ. Cette remise à zéro est exécutée à travers la valuation d'horloges $\theta[\lambda]$ définie par :

$$\theta[\lambda](h) = \begin{cases} 0 & \text{si } h \in \lambda \\ \theta(h) & \text{sinon} \end{cases}$$

$\theta[\lambda]$ représente en fait la projection orthogonale de la valuation d'horloges θ sur l'espace engendré par $H \backslash \lambda$.

Par ailleurs, nous notons $\varphi[\lambda]$ la contrainte d'horloges appartenant à $\Phi(H)$ et qui est satisfaite par tous les $\theta[\lambda]$ telles que θ appartient à la zone $[\varphi]$.

Exemple 3.2. *Considérant la contrainte* $\varphi = h_1 \geq 1 \wedge h_1 \leq 3 \wedge h_1 - h2_{\leq} 1 \wedge h_1 - h2_{\geq} - 1$ $\wedge h_2 \geq 1 \wedge h_2 \leq 3$ *et* $\lambda = \{h_1\}$, *Nous obtenons* $\varphi[\lambda] = h_2 \geq 1 \wedge h_2 \leq 3$

3.2.4 Définition du modèle général

Définition 3.3. *Un graphe temporisé est un quintuplet* $A = (L_A, L_A^o, M_A, T_A, H_A)$ *:*

- L_A *est l'ensemble de places (locations),*
- $L_A^o \subset L_A$ *est l'ensemble des places initiales,*
- M_A *est un ensemble fini d'événements,*
- H_A *est un ensemble fini d'horloges,*
- $T_A \subset L_A \times M_A \times \Phi^+(H_A) \times 2^{H_A} \times L_A$ *est l'ensemble des transitions,*

Un graphe temporisé est une structure que nous allons utiliser dans la suite pour la définition de modèles d'automates temporisés.

3.3 Automate temporisé de Büchi

Un automate de Büchi [Büc62] est un STE muni d'un sous ensemble de ses états appelés ensemble d'états d'acceptation et qui caractérise ses traces infinies. Toute trace infinie d'un automate de Büchi doit visiter un nombre infini de fois son ensemble d'états d'acceptation.

Alur et Dill [AD90] ont été les premiers à définir un automate temporisé, c'est l'automate temporisé de Büchi.

Définition 3.4. *Un automate temporisé de Büchi est un sixtuplet* $A = (L_A, L_A^o, M_A, H_A, T_A, F_A)$ *où :*

- $(L_A, L_A^o, M_A, H_A, T_A)$ *est un graphe temporisé*
- $F_A \subset L_A$ *est le sous ensemble de places d'acceptation.*

La sémantique d'un automate temporisé de Büchi A est définie en lui associant le STE Σ_A dont les états sont des couples (s, θ) composés d'une place appartenant à L_A et d'une valuation d'horloges appartenant à $\Theta(H_A)$. Les transition de Σ_A sont de deux types :

- des transitions reproduisant l'écoulement du temps d'une durée $d \geq 0$ de la forme $(s, \theta) \xrightarrow{d} (s, \theta + d)$
- et des transitions instantanées de la forme $(s, \theta) \xrightarrow{m} (s', \theta[\lambda])$ telles que $(s, m, \varphi, \lambda, s')$ appartient T_A et θ satisfait φ.

De plus, les chemins dérivés de toutes les traces infinies de Σ_A doivent visiter un nombre infini de fois les places appartenant à F_A. Cette dernière condition garantit la progression. Les automates temporisés de Büchi n'ont pas une sémantique opérationnelle [Yov93] car une transition de Σ_A ne peut être exécutée que si elle fait partie d'une trace infinie qui visite un nombre infini de fois

les places appartenant à F_A. Ainsi décider de l'exécutabilité d'une transition de A ne peut pas être basé sur l'état courant.

Exemple 3.3. *Considérons l'automate temporisé de Büchi à la figure 3.2(a) avec* $\{s_2\}$ *comme ensemble d'états d'acceptation. La transition* $(s_1, 0) \xrightarrow{2.1} (s_1, 2.1)$ *, par exemple, n'est pas permise car l'automate ne va plus jamais quitter la place* s_1 *et les chemins infinis commençant par* $(s_1, 0) \xrightarrow{2.1} (s_1, 2.1)$ *ne passeront jamais par l'état d'acceptation* s_2*. Ainsi l'ensemble d'états d'acceptation représente une condition implicite pour forcer l'automate à quitter la place* s_1 *alors que la contrainte* $h \leq 2$ *est encore satisfaite.*

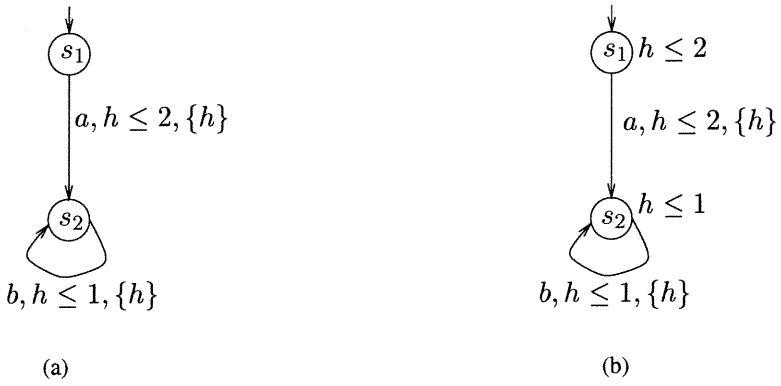


FIG. 3.2 – Exemple d'automates temporisés

3.4 Automate temporisé sécuritaire

Comme nous venons de le présenter à la section précédente, les automates temporisés de Büchi offrent un moyen théorique qui garantit le progrès à travers la définition d'un ensemble d'états d'acceptation. Les automates temporisés sécuritaires, introduits par Henzinger et al. [HNSY94, Alu99], utilisent un autre moyen plus pratique pour garantir le progrès et éviter le blocage. Les places d'un automate sécuritaire sont explicitement étiquetées avec une contrainte d'horloges appelée invariant ou condition d'activité de la place. L'invariant d'une place doit être satisfait lorsque l'automate entre dans cette place. L'invariant doit rester satisfait pendant tous le temps que l'automate passe dans cette place. Par conséquent, la place est quittée avant que son invariant ne devienne faux. Un exemple d'automate temporisé sécuritaire est montré à la figure 3.2(b). La contrainte

d'horloges $h \leq 2$ étiquetant la place s_1 représente son invariant. Sémantiquement, lorsque l'automate entre dans la place s_1 la contrainte $h \leq 2$ doit être satisfaite. La place s_1 doit être quittée avant l'expiration de son invariant $h \leq 2$.

Définition 3.5. *Un automate temporisé sécuritaire (ATS) est un graphe temporisé A auquel il est associé à chaque place s appartenant L_A une contrainte d'horloges appartenant à $\Phi(H_A)$ notées $Inv_A(s)$ et appelée invariant ou condition d'activité de la place s.*

De même, la sémantique d'un automate temporisé sécuritaire A est aussi définie en lui associant un STE Σ_A dont les transitions sont :

- des transitions reproduisant l'écoulement du temps d'une durée d de la forme $(s, \theta) \xrightarrow{d} (s, \theta + d)$, telle que pour tout réel positif d' inférieur ou égal à d, $\theta + d'$ satisfait $Inv_A(s)$
- et des transitions instantanées de la forme $(s, \theta) \xrightarrow{m} (s', \theta[\lambda])$ telles que $(s, m, \varphi, \lambda, s') \in T_A$, θ satisfait $\varphi \wedge Inv_A(s)$ et $\theta[\lambda]$ satisfait $Inv_A(s')$.

La relation de transition de Σ_A vérifie la propriété d'additivité des transition d'écoulement du temps. Pour tous d et d' deux réels positifs, si Σ_A comporte les transition $e \xrightarrow{d} e$ et $e' \xrightarrow{d'} e''$ alors $e \xrightarrow{d+d'} e''$ est aussi une transition de Σ_A.

Dans les systèmes de transitions classiques (discrets) nous parlons de traces pour représenter l'exécution du système. Pour les automates temporisés la notion d'état suivant n'a plus de sens car une transition $e \xrightarrow{d} e'$ de Σ_A représente le passage du système par tous les états $e + d$ avec $0 \leq d' \leq d$. Pour modéliser une exécution d'un système temporisé nous introduisons la notion de pas tel que définie dans [Yov93, HNSY94, ACD93].

Définition 3.6. *Soient A un ATS , Σ_A son système de transitions et e, e' deux états de Σ_A, nous disons que e' est accessible en un pas à partir de e, noté $e \triangleright e'$, s'il existe un réel $d \geq 0$ tel que $e \xrightarrow{d} e + d$ et $e + d \xrightarrow{m} e'$ avec $m \in M_A \cup \{0\}$. Nous le notons aussi $e \overset{d}{\triangleright} e'$ lorsque nous voulons mettre en évidence la durée d.*

Définition 3.7. *Soient A un ATS et Σ_A son système de transitions. Nous appelons une exécution σ de A toute séquence temporisée (finie ou infinie) de pas notée $\sigma = e_0 \overset{d_1}{\triangleright} e_1 \overset{d_2}{\triangleright} e_2 \overset{d_3}{\triangleright} \dots$*

Définition 3.8. *Pour chaque exécution $\sigma = e_0 \overset{d_1}{\triangleright} e_1 \overset{d_2}{\triangleright} e_2 \overset{d_3}{\triangleright} \dots$, nous définissons sa trace temporisée par la séquence $((\theta_1, l_1) \dots (\theta_j, l_j) \dots)$ telle que si l'on suppose que $e \overset{d}{\triangleright} e'$ est le $j$ème élément de σ tel que $e = (s, \theta)$ et $e + d \xrightarrow{m} e'$ et $m \in M_A$ alors $\theta_j \overset{def}{=} \theta + d$ et $l_j \overset{def}{=} m$.*

Remarque 3.1. *Une exécution d'un ATS est dite observable ssi sa trace temporisée est non vide.*

C'est le cas lorsque cette exécution n'est pas formée seulement par des transitions d'écoulement du temps. Il est à noter que les transitions d'écoulement du temps ne sont pas observables.

Définition 3.9. *Soient σ_1 et σ_2 deux exécutions d'un ATS A. Par définition nous supposons que $\sigma_1 = \sigma_2$ ssi σ_1 et σ_2 ont la même trace temporisée.*

3.5 Conclusion

Nous avons présenté la modélisation du temps dans les STEs. Ensuite nous avons présenté les modèles des automates temporisés de Büchi et les automates temporisés sécuritaires. Dans notre approche, nous avons choisi d'utiliser ce dernier modèle car il a une sémantique opérationnelle qui nous permet de transformer les scénarios en un automate temporisé sécuritaire.

Chapitre 4

État de l'art sur les approches de spécification par intégration de scénarios

Nous consacrons ce chapitre à une revue de l'état de l'art concernant l'utilisation des scénarios pour la spécification d'un système. Sachant que les scénarios représentent des parties de la spécification, leur intégration consiste à les regrouper en une seule spécification globale. Certains auteurs l'appellent par *composition de scénarios*. Cette appellation peut porter à confusion dans la mesure où cette terminologie est déjà utilisée dans la théorie des automates. Nous faisons référence à la composition parallèle des STEs par exemple. Cependant dans certain cas [Lus97], l'intégration de scénarios peut représenter leur composition parallèle.

Les caractéristiques d'une méthode d'intégration de scénarios sont principalement :

- l'acquisition et la représentation des scénarios
- l'intégration des scénarios en une spécification
- l'analyse et la vérification des scénarios et de la spécification résultante de leur intégration
- l'utilisation des techniques formelles dans les trois points précédents

Dans le plan du chapitre, la section 4.1 décrit différents travaux basés sur l'intégration des scénarios pour la spécification des systèmes. Pour chaque approche nous nous intéressons aux quatre caractéristiques que nous venons d'énumérer. La section 4.2 est consacrée à une étude comparative entre les approches étudiées.

4.1 Présentation de l'état de l'art

4.1.1 Approche de Hsia et al.

Cette approche [HSG+94] propose une méthodologie pour la spécification des exigences des usagers basée sur les scénarios. Elle vise à fournir une spécification formelle et complète qui sera utilisée pour la conception et le test. Le processus de spécification se compose de plusieurs phases :

Extraction des scénarios :

Durant cette étape un analyste établit les scénarios requis en consultation avec l'usager. Pour chaque vue d'usager les scénarios sont constitués sous forme d'un arbre représenté à la figure 4.1(a). Une vue d'usager représente un regroupement d'usagers qui voient le comportement du système de la même manière. Un scénario est déclenché par un agent et il est constitué d'une séquence d'événements. Chaque événement change l'état du système et peut en déclencher à son tour un autre.

L'arbre des scénarios est construit à partir de l'état initial (racine) en ajoutant des arcs étiquetés par chaque événement possible à cet état vers de nouveaux noeuds représentant l'état du système après l'événement. Ensuite, le même traitement est répété pour chaque nouvel état.

Formalisation des scénarios :

Les scénarios sont transformés sous forme d'une grammaire régulière. L'ensemble des grammaires constituées sera assemblé sous forme d'un automate à états finis dans lequel les noeuds sont les états du système et les transitions sont les événements. Cet automate trace le comportement du système selon la vue d'un usager (figure 4.1(c)).

Vérification des scénarios :

La vérification se fait en utilisant la grammaire et l'automate à états finis précédents. Cette double vérification permet d'assurer que l'automate construit est conforme à la grammaire. Les propriétés à vérifier sont :

- que chaque état est accessible à partir de l'état initial et inversement que l'état initial est accessible à partir de tous les états,
- qu'il n'y a pas de non déterminisme,
- et qu'il n'y a pas de transition spontanée sans aucun événement.

Génération des scénarios :

Les scénarios sont générés à partir de l'automate. Ils représentent les traces de l'automate

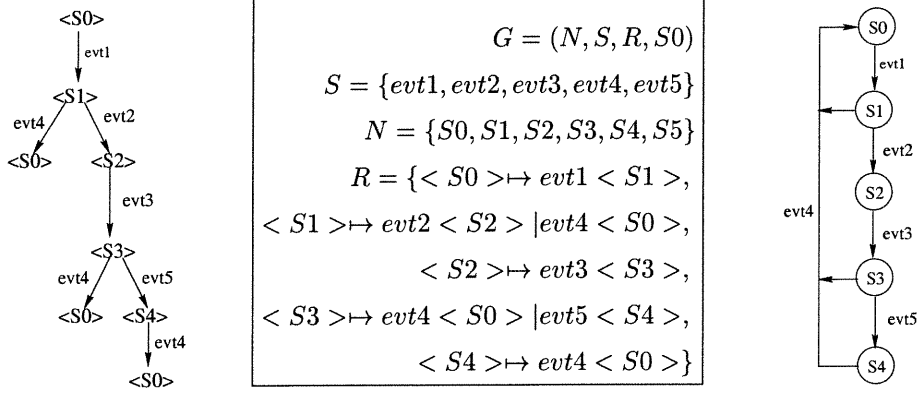


FIG. 4.1 – Différentes étapes dans l'approche de Hsia et al.

qui commencent et se terminent par l'état initial. Ces scénarios doivent être validés par l'usager.

Validation des scénarios :

Les utilisateurs exécutent les scénarios générés et vérifient que le comportement du système répond à leurs attentes. D'abord les scénarios d'une seule "vue utilisateur" sont exécutés d'une manière indépendante. Ensuite les scénarios de vues distinctes sont simultanément lancés afin de détecter l'existence d'interactions indésirables.

L'arbre des scénarios construit pour une vue est complet par construction, car à chaque noeud, tous les événements possibles sont ajoutés. La vérification des scénarios se fait par vue d'usager et ne tient pas compte des interactions entre vues qui peuvent mener à des blocages.

4.1.2 Approche de Koskimies et al.

Koskimies et al. [KM94] proposent un algorithme pour la génération d'une machine à états décrivant le comportement dynamique d'un objet dans la méthode OMT [RBP+91] à partir de diagrammes d'interactions aussi appelés scénario ou use cases. Selon la terminologie d'OMT, un scénario est une trace qui décrit l'ordre d'échanges d'événements entre objets. Une machine à états

est un graphe dirigé où les noeuds sont les états et les transitions sont des événements contraints par des "guards". Un état, auquel une action peut être associée, est une abstraction des valeurs des attributs d'un objet.

La génération de la machine à états se fait par induction en se basant sur l'algorithme de Biermann [BK76]. Celui-ci consiste à construire un programme à partir d'un ensemble de traces données comme une séquence complète d'instructions et de conditions de la forme (i, c). Le programme construit est une machine à états dans laquelle les transitions sont de la forme (p, c, q) où p, q sont des instructions étiquetées et où c est une contrainte sur l'état de la mémoire. L'algorithme infère un programme déterministe et *complet*. Un programme est dit *complet* si :

1. il contient une seule transition qui part de l'instruction initiale
2. et s'il comporte une transition partant de l'instruction p alors l'ensemble formé par toutes les conditions des transitions partant de p est complet, c'est à dire qu'il existe toujours une transition partant de p dont la condition qui est vraie.

L'algorithme de Biermann produit un programme déterministe minimal en nombre d'états. Son principe consiste à attribuer un étiquetage minimal aux instructions dans les états. Au départ, l'algorithme commence par un étiquetage minimal qui associe une seule étiquette par type d'instructions et qui sera augmenté après chaque itération jusqu'à l'obtention d'un programme complet et déterministe.

Pour appliquer l'algorithme de Biermann une correspondance est définie entre les scénarios représentés sous forme de diagrammes d'événements et les traces d'instructions d'un programme. Soit O un objet pour lequel on veut construire une machine à états décrivant son comportement. A partir de chaque scénario Sc où l'objet O interagit, on extrait une séquence d'événements $S_{Sc} = ((e_{e1}, e_{r1}), \ldots, (e_{en}, e_{rn}))$ où e_{ei} est un événement envoyé par O et e_{ri} est l'événement de réception intercepté par O après e_{ei}. Deux événements spéciaux $null$ et $default$ peuvent être introduits pour avoir une séquence sous forme d'*envoi/réception*. L'algorithme de Biermann s'applique en considérant que les événements d'envoi comme des actions (des instructions) et ceux de réception comme des conditions. La machine à états synthétisée accepte au moins (et exactement si les scénarios sont complets) toutes les traces S_{Sc} pour tous les scénarios où O participe.

L'algorithme de Biermann a une complexité exponentielle et il est sensible à l'ordre d'entrée des traces mais garantit toujours une machine à états déterministe minimale. Mais, à cause du manque de sémantique dans les actions, les scénarios risquent d'être combinés d'une manière indésirable. De plus la condition de complétude nécessite qu'il y ait une transition pour chaque événement et à partir de chaque état, ce qui est difficilement réalisable en pratique.

4.1.3 Approche de Glinz

Glinz [Gli95] utilise les Statecharts [Har87] pour la représentation et l'intégration des scénarios. Les Statecharts (cf. section 2.3) sont une variété des FSMs qui permettent la décomposition sous forme d'une hiérarchie de Statecharts et permettent la concurrence à travers des Statecharts parallèles.

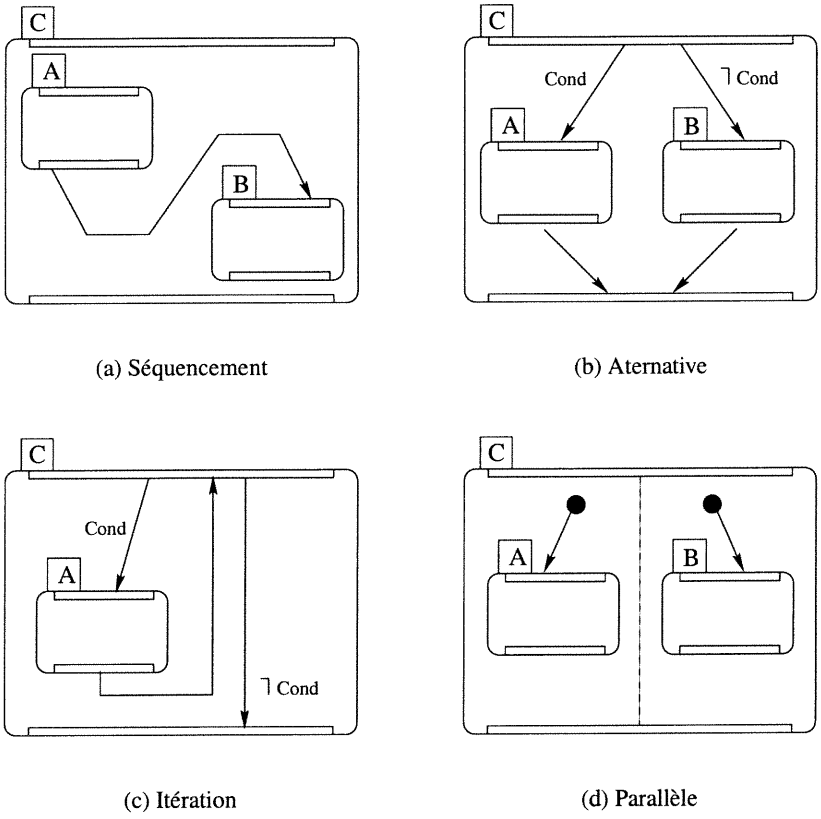
Un scénario est représenté sous forme d'un Statechart. Les scénarios sont intégrés d'une manière explicite avec des opérateurs sous forme de patrons d'intégration qui indique leur ordre d'exécution (figure 4.2). L'intégration explicite de scénarios consiste à imposer un schéma d'intégration explicite qui lie les scénarios les uns aux autres dans la spécification à générer. Après leur intégration, les scénarios restent visibles dans le Statechart composé. Ceci impose que les scénarios soient disjoints et sans chevauchement. Autrement, les scénarios seront éclatés manuellement en scénarios disjoints avant l'intégration selon Glinz.

Le modèle Statechart des scénarios intégrés est traduit en un automate à état fini qui sert à la vérification du non blocage, à l'analyse d'accessibilité et à la simulation.

4.1.4 Approche d'Amyot et al.

Dans l'approche d'Amyot et al. [ALB97], un scénario est considéré comme étant une séquence d'événements et d'actions qui sont reliés d'une façon causale dans le but d'offrir une fonctionnalité du système. Ces scénarios sont représentés dans le formalisme des *use cases map* (UCM) [BC95].

L'idée de base des UCMs consiste à relier d'une manière causale des responsabilités (des événements et/ou des actions) par un chemin qui traverse des composantes organisationnelles établies dès les premières phases de l'acquisition des besoins. Les UCMs sont décrits par trois éléments indépendants : les chemins, les composantes et les responsabilités ; comme c'est illustré



(a) Séquencement (b) Aternative

(c) Itération (d) Parallèle

FIG. 4.2 – Patrons d'intégration [Gli95]

dans la figure 4.3. Ces éléments sont combinés pour montrer seulement les séquences causales entre les responsabilités. Ainsi les UCMs ne comportent aucune information sur le mode de communication entre composantes, ni comment leurs états internes contrôlent les séquences de communication, échangent ou stockent les données [Buh98].

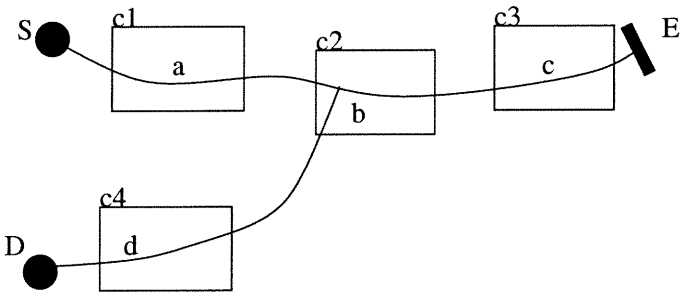


FIG. 4.3 – Un exemple d'UCM

Cet UCM montre deux flux d'exécutions possibles : l'un part de S à E à travers les composantes c1,c2 et c3, et l'autre de D à E en passant par c4,c2 et c3.

Une fois les scénarios exprimés sous forme d'UCMs, il est possible de considérer différentes alternatives architecturales pour le système à concevoir. Des contraintes dues à la présence de composantes obligatoires, déjà existantes ou dictées par d'autres considérations de limitation de ressources ou de performance peuvent guider le choix d'une architecture. En tenant compte des scénarios établis, cette architecture est projetée sur les UCMs en regroupant plusieurs composantes dans une boîte (figure 4.4).

A cette étape de la conception, l'analyste traduit l'ensemble des UCMs disponibles avec l'architecture choisie en une spécification LOTOS. Cette spécification permet les mêmes traces d'événements et d'actions que les UCMs et admet la même structure architecturale.
Par ailleurs, la spécification LOTOS comporte d'autres types d'information au niveau de la spécification des paramètres et des types abstraits de données en plus des ports de communication et des types de messages qui sont ajoutés par la suite.

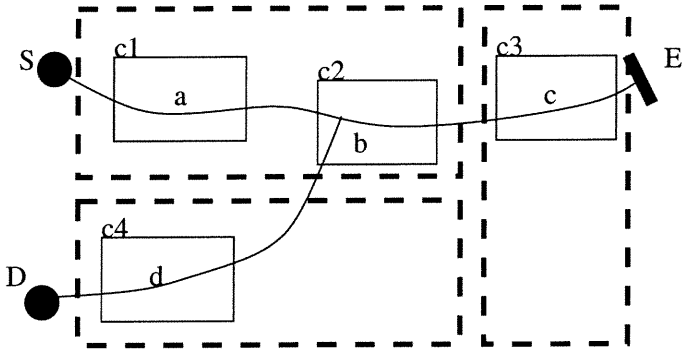


FIG. 4.4 – Projection d'une architecture à l'UCM de la figure 4.3

Enfin, des tests fonctionnels sont directement générés à partir des scénarios exprimés sous forme d'UCMs pour valider la conformité du prototype LOTOS manuellement synthétisé. Cependant, si la génération de la spécification était faite de manière automatique, le test de conformité du prototype par rapport à la spécification sous forme d'UCMs aurait pu être évité.

4.1.5 Approche de Somé et al.

Somé et al. [SDV95, SD96, SDV96a, SDV96b, Som97] proposent une méthodologie pour l'acquisition des exigences basées sur les scénarios. Cette phase constitue une phase critique où des erreurs peuvent s'introduire lors de la conversion des besoins en spécification, sans oublier que ces exigences risquent dès le départ de contenir des incohérences ou des incomplétudes. Cette approche permet la modélisation des systèmes temps-réels et des systèmes réactifs dont les comportements sont contraints par l'aspect temporel et par leurs interactions avec leur environnement.

Un scénario est une description partielle du comportement du système et de son environnement dans une situation restreinte selon l'intérêt de l'analyste ou selon un point de vue d'un usager (ou un expert). Les besoins sont exprimés sous forme de scénarios qui sont intégrés pour dériver d'une manière automatique une spécification exprimée sous forme d'un automate temporisé modélisant le comportement global du système.

Cette approche distingue quatre phases itératives :

Description du système : Elle permet d'énumérer les composantes du système ainsi que les types

de leurs attributs, et décrit également les différentes opérations que les composantes exécutent. Ces opérations font changer l'état du système. L'exécution d'une opération entraîne la suppression de certaines conditions antérieures et implique l'ajout d'autres pour tenir compte du changement dans l'état.

Acquisition des scénarios : Les scénarios sont exprimés dans un langage restreint constitué de phrases en langage naturel restreint relié par des structures conditionnelles (`WHEN` et `IF ... THEN ...`) et des opérateurs temporels (`BEFORE`, `AFTER`, `AT`, etc ...).
Un exemple de scénario est donné à la figure 4.5. Le mot clé `WHEN` permet de définir les conditions de l'état initial du scénario. Ensuite le scénario est exprimé sous forme d'une séquence d'interactions. Une interaction est un couple $< stimuluss/reaction >$ étiquetant une transition gardée par une contrainte temporelle. De même une durée maximale peut être affectée à un scénario avec la structure `DELAY`.

```
Sc1 WHEN ATM display is card_insert_prompt
        IF USER inserts card THEN ATM displays pin_enter_prompt
        BEFORE 5 sec IF USER enters personal_identification_number
           THEN ATM checks USER identification
           INTERACTION DELAY 60 sec
           ON EXPIRY ATM ejects card AND ATM reinits
        IF USER identification is valid
           THEN ATM displays operation_menu
        IF USER selects cash_withdrawal
           THEN ATM asks amount
    DELAY 60 sec ON EXPIRY ATM ejects card AND ATM reinits
```

FIG. 4.5 – Exemple de scénario décrivant les interactions avec un "Automatic Teller Machine" (ATM) [Som97]

Algorithme d'intégration des scénarios : Un scénario est considéré comme une trace temporisée d'un automate à construire. L'algorithme d'intégration prend en entrée un scénario et un automate temporisé (possiblement vide au départ) et retourne un automate temporisé qui

accepte toutes les traces de l'automate initial en plus de celle du scénario ajouté. La figure 4.6 montre l'automate construit à partir du scénario de la figure 4.5. L'automate temporisé résultant est en fait une FSM étendue par les horloges.

Détection d'incohérences et d'incomplétude : L'automate temporisé résultant de l'étape précédente est analysé en vue de détecter certaines incohérences temporelles à l'aide d'un ensemble de règles d'inférence qui font la propagation des contraintes temporelles.
Par ailleurs, le traitement de l'incomplétude permet de combler certains oublis de spécification et inciter l'analyste à ajouter des scénarios pour décrire le comportement du système face à certains stimuli qui sont décrits dans la description du système mais non utilisés dans les scénarios.

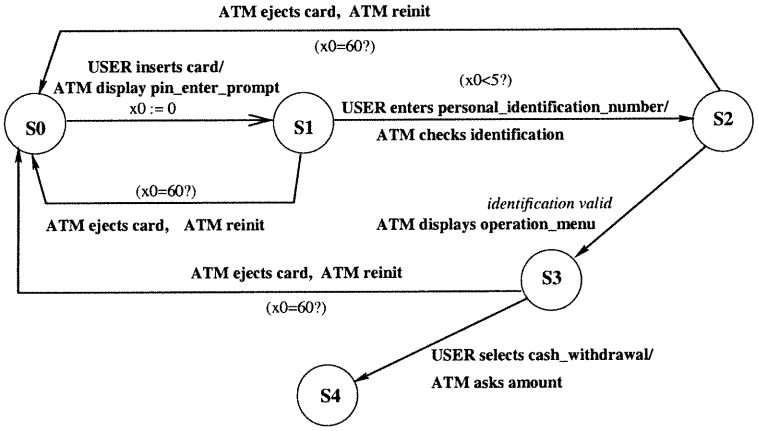


FIG. 4.6 – Automate temporisé représentant le comportement du scénario de la figure 4.5.

Ces tâches sont itératives dans la mesure où à chaque phase on peut retourner aux phases précédentes pour compléter la description du système, ajouter ou modifier des scénarios.

4.1.6 Approche de Dano et al.

Dans leur approche [Dan96, DBB97, Dan97], Dano et al. utilisent les scénarios dans leurs activités d'acquisition des besoins et de spécification orientée objet afin de faciliter la communication entre les experts du domaine et l'analyste durant cette phase et de produire une spécification formelle du système.
Pour atteindre ces objectifs, deux techniques de description des scénarios ont été proposées.

D'abord une représentation des scénarios sous forme de tables pour la communication avec les experts du domaine. Ensuite une représentation des scénarios sous forme de réseaux de Pétri colorés synchronisés est établie par l'analyste.

La spécification des besoins du point de vue des experts du domaine est stockée dans deux tables appelées "*Tables Partielles d'un SCénario*" (TPSC) :

- la première contient un ensemble de fonctions séquentielles qui sont exécutées par un acteur.
- la seconde table contient les états des types d'objets durant chaque fonction listée dans la première. Les types d'objets sont ceux définis dans la spécification statique et qui sont pertinents pour le scénario considéré. La spécification statique utilise les notations de la méthode OMT [RBP+91].

Chaque TPSC ne comporte qu'une description partielle d'un scénario. L'ensemble des TPSC nécessaires pour décrire un scénario sont regroupées sous forme d'un arbre appelé "*Arbre d'un SCénario*" (ASC). Chaque noeud d'un ASC est composé de deux TPSC et étiqueté par les conditions prises en compte lors de la construction des deux TPSC. La racine d'un ASC est composée de deux TPSC initiales.

L'autre représentation des scénarios est une représentation sous forme d'un réseau de Pétri. Le passage entre les TPSCs aux réseaux de Pétri n'est pas simple. L'analyste est aidé par certaines règles facilitant la construction d'un réseau de Pétri pour chaque scénario à partir de TPSC. Un tel réseau de Pétri est appelé "*Réseau de Pétri de SCénario*" (RDPSC).

La description des scénarios permet de n'avoir qu'une description partielle de l'application. Ainsi, l'utilisation d'outils de simulation des réseaux de Pétri appliqués aux RDPSC construits mettrait en évidence des places qui ne pourraient jamais être atteintes (jamais marquées), et des blocages. De tels cas supposeraient que d'autres scénarios ont dû être exécutés avant ou seront exécutés par la suite.

Afin d'obtenir une description globale du système, la notion de "*liens temporels entre les scénarios*" a été introduite. Les différents liens temporels possibles entre deux scénarios sc1 et sc2 (cf. figure 4.7) sont définis à partir des sept relations pouvant exister entre deux intervalles temporels [All83]. On entend ici par intervalle temporel, la durée d'un scénario. Ensuite les différentes

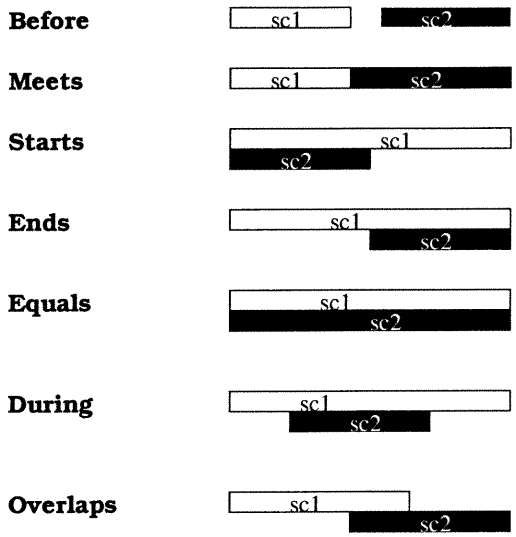


FIG. 4.7 – Ordonnancement entre les scénarios

possibilités d'ordonnancement des scénarios sont représentées dans un diagramme.

A partir des spécifications RDPSC l'analyste construit pour chaque objet un diagramme de transition d'états comme ceux définis dans la méthode OMT.

Les liens temporels entre les scénarios n'ont pour sémantique que de montrer l'ordonnancement entre les scénarios. Cette notion est différente de l'intégration des scénarios au sens des traces dans la mesure où ces scénarios, tels que construits, sont supposés indépendants et sans chevauchement d'actions lors de leurs exécutions. Le chevauchement dont il s'agit ici est seulement temporel. Ceci ne permet pas de détecter certains comportements indésirables causés par les interactions entre les scénarios.

4.1.7 Approche de Desharnais et al.

Dans cette approche [DKFM97], les scénarios sont exprimés par des formules en terme de relations. Une relation est définie comme une sous-partie du produit cartésien d'un certain nombre d'ensembles. Un scénario est formellement représenté par un triplet (T, R_e, R_s) où T est l'espace des états, il est donné comme le produit cartésien des domaines des variables. R_e

et R_s sont deux relations disjointes de T. R_e représente les actions déclenchées par l'environnement considéré comme un système non contraint (un utilisateur humain) et R_s sont les réactions du système. Le formalisme utilisé est très proche du langage Z [Spi88]. L'intégration de deux scénarios (T, R_e, R_s) et (T', R'_e, R'_s) donne lieu à un nouveau scénario $(\mathcal{T}, \mathcal{R}_e, \mathcal{R}_s)$ qui est défini par un opérateur relationnel sous forme d'une formule relationnelle. Cependant la relation de conformité entre le scénario $(\mathcal{T}, \mathcal{R}_e, \mathcal{R}_s)$ et le scénario (T, R_e, R_s) n'est pas formellement définie ; en d'autre termes comment le scénario $(\mathcal{T}, \mathcal{R}_e, \mathcal{R}_s)$ assume le comportement de l'un des scénarios (T, R_e, R_s) qu'il intègre.

L'un des objectifs de l'utilisation des scénarios est de faciliter la communication entre l'usager et l'analyste à travers un médium formel. Ce but risque d'être difficile à atteindre avec ce formalisme car il ne peut pas être facilement lisible par l'usager.

4.1.8 Approche de Lustman

Lustman dans [Lus97], propose une méthode pour la génération de la spécification d'un système en utilisant l'approche scénario. Cette méthode combine des tâches formelles et d'autres informelles pour produire une spécification sous forme d'un automate à partir de la composition de plusieurs scénarios automates en un seul. L'analyse de la spécification permet de détecter certaines erreurs et de proposer d'éventuelles corrections

Dans cette approche, un scénario est un automate qui décrit le comportement d'une composante où les transitions sont étiquetées par les actions exécutées par la composante. Les états de cet automate portent des labels représentant les états de la composante. La sémantique des labels des états et des transitions est établie manuellement par l'analyste.

L'intégration de deux scénarios est effectuée en deux étapes :

- A partir des automates des scénarios on construit automatiquement leur automate composé tel que illustré à la figure 4.8. Il est à noter que les alphabets des deux automates scénarios sont disjoints et donc il n'y a encore aucune synchronisation entre eux.
- La deuxième phase consiste à valider l'automate composé en éliminant certains états et transitions selon la sémantique définie par l'analyse pendant l'étiquetage des états et des transitions des scénarios automates.

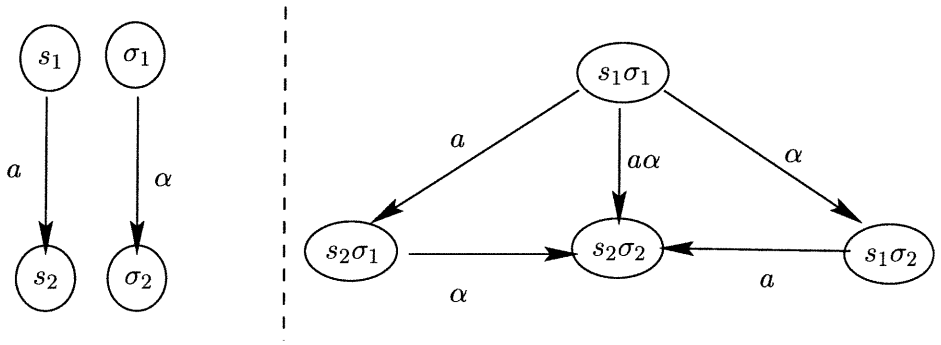


FIG. 4.8 – Un exmple illustrant la composition des automates scénarios dans [Lus97]

Ensuite, le processus se poursuit par l'analyse de l'automate composé et validé, ce qui permet la détection et la correction de trois types d'erreurs :

- L'élimination de certains états et transitions pendant la validation peut rendre certains états inaccessibles. Pour corriger cette situation, certains états et transitions éliminés pendant la validation ne devaient pas l'être.
- Le deuxième type d'erreurs correspond à des inconsistances dans les scénarios intégrés. Soient A_1 et A_2 deux scénarios automates et A leur automate composé validé. Pour détecter ce type d'erreurs on compare A_1 (respectivement A_2) avec la projection de l'automate composé validé A sur A_1 (respectivement A_2).
- Le troisième type d'erreurs se manifeste par un manque au niveau de l'automate composé car les deux scénarios sont cohérents mais incomplets. Ce manque peut être comblé en complétant les deux scénarios.

4.1.9 Approche d'Elkoutbi et al.

Cette approche [EK98] vise à produire la spécification d'un système interactif sous forme d'un réseau de Pétri coloré. L'activité de modélisation comprend deux niveaux d'abstraction : le niveau des *use cases* qui correspond au niveau des use cases tel que défini dans la méthode de conception orienté objet UML (*Unified Modeling Language*) [UML97, BJR97] et le niveau de scénarios qui est un raffinement du premier.

Un *use case* est défini comme étant une description générique de toute une transaction par contre un scénario représente une exécution possible d'un use case. Ainsi un use case peut avoir plusieurs scénarios d'exécution. L'activité de modélisation comprend deux processus :

1. En se basant sur UML, un *use case diagram* est établi en premier lieu. Ce diagramme montre la relation des différents use cases entre eux et avec les acteurs extérieurs. Ensuite ce diagramme est transformé en un réseau de Pétri qui tient compte des ordonnancements des use cases.
2. Le deuxième processus qui vise à produire pour chaque use case un modèle sous forme d'un réseau de Pétri coloré qui englobe tous les scénarios du use case. Ce processus comprend plusieurs tâches qui sont appliquées sur chaque use case :
   (a) Acquisition des scénarios d'un use case sous forme d'un diagramme de séquences qui est un simple MSC qui ne comporte que les acteurs comme instances de processus et les messages qu'ils échangent.
   (b) Établissement d'une table d'états des objets dérivée manuellement à partir des diagrammes de séquences. Cette tâche consiste à étiqueter les états du système après chaque envoi et réception d'un message de tel sorte à avoir un étiquetage uniforme dans tous les scénarios du use case. Chaque état est caractérisé par l'ensemble des attributs des objets du système qui participe au scénario.
   (c) Dans cette étape le scénario est représenté par un réseau de Pétri coloré où les places sont des états de la table établie dans l'étape précédente et où les gardes des transitions sont les messages
   (d) Les scénarios d'un use case représentés sous forme de réseaux de Pétri colorés sont intégrés en un seul réseau de Pétri global du use case. L'algorithme d'intégration génère un réseau de Pétri coloré qui permet exactement les comportements décrits dans les scénarios intégrés en utilisant une couleur de jeton différente pour chaque scénario. Par exemple la figure 4.9 montre deux scénario sc1 et sc2 qui sont intégrés pour donner le réseau de Pétri coloré sc. A l'aide des couleurs, le réseau de Pétri sc ne permet d'avoir le chemin (s1,s2,s3,s7,s5) comme exécution du use case car il ne représente pas un scénario intégré.
3. On remplace chaque use case par son réseau de Pétri (intégrant tous ses scénarios) de la phase 2.(d) dans le réseau de Pétri initial obtenu à la fin de 1.

Cette approche suppose implicitement qu'il n'y a pas de chevauchement entre les use cases établis dans le réseau de Pétri résultant de l'étape 1.

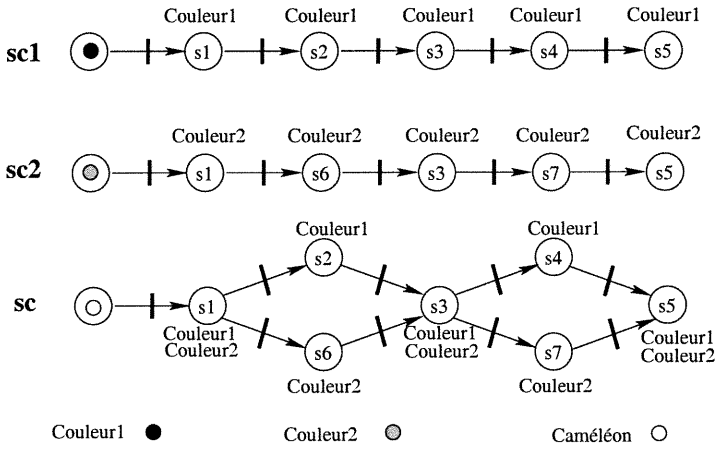


FIG. 4.9 – Intégration des scénarios sous forme d'un réseau de Pétri coloré

Le réseau de Pétri coloré sc intègre exactement le comportement de sc1 et de sc2. Le jeton caméléon dans sc devient soit de couleur1 soit de couleur2 en franchissant la première transition. Le comportement suivra alors soit sc1 soit sc2.

4.2 Étude comparative des différentes approches

Dans les méthodes de conception orientées objet telles que OMT [RBP+91] et UML [UML97], un scénario représente une exécution possible d'un *use case*. Dans ce cas, un scénario est lié à un *use case* qui est une description d'une fonctionnalité du système. Mais d'une manière générale, un scénario peut être défini comme une séquence d'interactions entre une composante (aussi appelée objet ou encore agent) et son environnement qui est formé par les autres composantes et l'environnement externe du système. Un scénario ne décrit alors qu'un comportement partiel du système.

Cette définition fait l'unanimité des approches scénario précédemment présentées. Sauf que certains auteurs ajoutent qu'un scénario est défini selon un but qui vise à réaliser une fonctionnalité

du système [EK98, DBB97, KM94, ALB97, Lus97], par contre d'autres auteurs laissent aux utilisateurs et aux analystes la liberté d'exprimer leurs scénarios selon toutes autres considérations [Som97, HSG+94, Gli95].

Par ailleurs, dans toutes les méthodes évoquées ci-haut, les scénarios sont utilisés dans le but d'établir une spécification cohérente et conforme aux exigences des utilisateurs. Pour arriver à ce but plusieurs étapes sont nécessaires. Les étapes les plus communes à toutes les approches sont :

- La représentation des scénarios.
- L'intégration des scénarios en une spécification.
- L'analyse des scénarios et de la spécification résultante de leur intégration.

La suite de cette section fera un développement de ces trois étapes.

4.2.1 La représentation des scénarios

La représentation des scénarios consiste à transformer les exigences des utilisateurs récoltées sous forme de documents en langue naturelle vers une représentation sous forme de scénarios. Souvent cette représentation est formelle. Plusieurs représentations ont été utilisées parmi lesquelles, nous citons :

- les langues naturelles restreintes [Som97],
- les MSCs [EK98, KM94]
- les tables de description de comportement [DBB97],
- les Réseaux de Pétri [DBB97, EK98]),
- les statecharts [Gli95],
- les traces d'automate [Som97, KM94]
- et les automates à états finis [Lus97].

Certaines approches sont citées dans plus d'un formalisme de représentation des scénarios. En fait ces approches supportent deux représentations pour les scénarios :

- une représentation qui sert pour la communication avec l'utilisateurs. Cette représentation est moins formelle mais elle a l'avantage d'être plus lisible par les utilisateurs pendant la validation des scénarios.
- une représentation plus formelle qui supporte le traitement automatique lors de l'intégration des scénarios et de l'analyse.

4.2.2 Les algorithmes d'intégration des scénarios

L'intégration des scénarios consiste à en regrouper plusieurs sous forme d'un seul modèle qui permet tous les comportements spécifiés dans les scénarios. La sémantique donnée à l'intégration varie d'une approche à une autre. Notre étude nous a permis de distinguer les sémantiques suivantes :

Composition parallèle : Utilisée dans [Lus97] où on entend par composition de deux scénarios le modèle défini par leur composition parallèle qui donne la synchronisation entre composantes. Dans cette approche, chaque scénario décrit le comportement d'une composante.

Intégration pendant l'acquisition des scénarios : Cette méthode est pratiquée dans [HSG+94] (section 4.1.1) où les scénarios sont récoltés sous forme d'un arbre. A chaque noeud de l'arbre toutes les actions possibles sont considérées. Un scénario sera tout simplement un chemin de l'arbre des scénarios.

Ordonnancement : Ce type d'intégration consiste à donner un ordonnancement reliant l'exécution des scénarios [DBB97, Gli95].

Intégration au sens de traces : Ce type d'intégration consiste à synthétiser un modèle qui accepte les traces données. Ces traces représentent les scénarios [KM94, Som97]. Cette intégration est dite implicite dans le cas où aucune contrainte n'est imposée sur l'ordonnancement des traces. Dans le cas où des contraintes d'ordonnancement des traces sont spécifiées, on parle d'intégration explicite ou déclarative.

Pour les deux derniers types d'intégration, on peut se poser la question suivante : Est-ce que le modèle résultant de l'intégration des scénarios comporte exactement les mêmes scénarios ou permet-il d'autres scénarios non initialement donnés ? Pour répondre à cette question, il faudrait définir pour chaque méthode d'intégration considérée, une relation formelle de conformité qui relie la spécification résultante de l'intégration et ses scénarios.

La réponse à la question précédente dépend de l'approche. Dans le cas de l'approche d'Elkoutbi et al., elle interdise dans l'intégration d'avoir des scénarios non explicitement donnés. Nous pou-

vons justifier cela pour le cas de [EK98] par le fait que les scénarios intégrés représentent toutes les exécutions possibles d'un *use case*.

Par contre, d'autres auteurs permettent au modèle de l'intégration d'accepter des scénarios qui n'étaient pas donnés lors de l'intégration. Cette situation arrive lorsque deux scénarios sc1 et sc2 ont une portion commune telle que sc1=(a1,a2,a3) et sc2=(a4,a2,a5) par exemple. Le scénario sc3=(a1,a2,a5) peut être permis par le modèle qui intègre sc1 et sc2. C'est le cas pour les approches de Koskimies et Somé. Dans notre approche cette possibilités est permise mais nous pouvons aussi imposer, selon le désir du concepteur, que le résultat de l'intégration des scénarios ne comporte que les scénarios de départ.

4.2.3 Analyse des scénarios et de la spécification

Après l'acquisition des scénarios et leur intégration, le processus se poursuit par les activités de validation et de vérification des scénarios et de la spécification générée.

La validation

La validation consiste à soumettre aux utilisateurs les scénarios acquis ou la spécification générée pour qu'ils se prononcent sur leur conformité par rapport aux exigences. Le formalisme de description des scénarios utilisé doit être facilement compréhensible par les utilisateurs.

Parmi les approches utilisant la validation au niveau scénarios on trouve [HSG+94, DBB97]. Pourtant la facilité de la validation dans les approches scénarios constitue l'une des principales raisons de leur utilisation. En effet, le comportement décrit dans un scénario, étant restreint et partiel, l'utilisateur peut facilement le comprendre et le valider. Par ailleurs, la validation est aussi utilisée au niveau de la spécification constituée par l'intégration des scénarios [Lus97].

La vérification

La vérification dans les approches scénarios s'effectue aussi bien sur les scénarios avant leur intégration qu'après en utilisant le modèle qui en résulte. La vérification d'un scénario sert à s'assurer que le scénario est conforme à la description statique du système [Som97].

Par contre la vérification du modèle résultant de l'intégration des scénarios vise à détecter des incohérences qui peuvent exister entre les scénarios puisque ces derniers ne spécifient que des comportements partiels et qui peuvent être dictés par des utilisateurs différents. Ainsi, l'existence de contradiction est très probable. Les techniques utilisées pour la vérification des modèles intégrant les scénarios sont :

- Analyse d'accessibilité
- Simulation
- Vérification de la complétude et du non déterminisme.

L'analyse d'accessibilité permet de vérifier si tout état est accessible à partir de l'état initial. Par conséquent, la spécification ne comporte pas de blocage et tous les scénarios peuvent s'exécuter. En effet, à cause d'une mauvaise interaction avec un autre scénario, un comportement décrit dans un scénario peut ne pas être possible.

Dans certains formalismes, l'analyse d'accessibilité est très coûteuse ou même non décidable. Le recours à la simulation permet d'exercer des comportements de la spécification afin de détecter des interblocages ou de vérifier des situations spécifiques. Pendant la simulation, l'analyste prend la spécification d'une composante et joue le rôle de son environnement. L'analyste fournit les stimulations et vérifie la réaction de la composante. L'inconvénient de la simulation est qu'elle est non exhaustive.

La vérification de la complétude consiste à vérifier qu'à chaque état du système toutes les actions possibles sont présentes dans la spécification. Cette opération sert à éliminer les oublis et à traiter certains cas d'erreurs. Cependant, ce procédé ne permet pas de garantir que la spécification est complète.

Le traitement du non déterminisme consiste à le détecter puis à vérifier s'il correspond à une erreur de spécification ou bien s'il est causé par une abstraction. Dans le premier cas il est corrigé mais dans le second il est accepté dans l'attente d'un raffinement qui va l'éliminer.

4.3 Conclusion

Dans ce chapitre nous avons présenté les travaux reliés à la spécification d'un système basés sur l'intégration des scénarios. Nous avons aussi présenté une synthèse qui compare les différents travaux étudiés.

Ce chapitre marque la fin de la première partie de cette thèse. Dans le chapitre suivant nous allons présenter les fondements théoriques de notre approche d'intégration des scénarios pour la génération d'une spécification d'un système réactif temps-réel.

Chapitre 5

Fondement de notre approche pour la génération d'une spécification d'un système réactif temps-réel

Ce chapitre présente la base de notre approche qui consiste à synthétiser un automate temporisé à partir d'un ensemble de spécifications partielles du comportement d'un système réactif temps-réel.

La section 5.1 décrit les éléments caractérisant un système réactif temps-réel. La formalisation des caractéristiques d'un tel système est basée sur la définition des variables discrètes et des horloges qui représentent à chaque instant son état. La spécification du comportement du système décrit l'évolution de son état en fonction des événements observés et de l'écoulement du temps. Il faut de plus spécifier aussi l'ensemble des événements que le système peut observer. Pour décrire le comportement du système, nous utilisons une forme canonique et élémentaire que nous appelons *action-règle* décrite à la section 5.2. La sémantique d'un ensemble d'actions-règles est donnée par un système de transitions auquel nous associons un ATS. La section 5.3 traite la réduction de cet ATS afin de minimiser le risque d'une explosion combinatoire du nombre de ses places. Enfin, la section 5.4 illustre notre approche par un exemple d'application.

Dans le reste de la thèse, le terme *système* désigne le système réactif temps-réel qui est en cours de spécification.

5.1 Éléments de la description d'un système réactif temps-réel

Un système *réactif temps-réel* a besoin d'au moins deux éléments pour décrire son comportement. Le premier concerne l'attribut *réactif* pour lequel nous spécifions un ensemble d'événements auxquels le système réagit. Le second élément est relié à l'aspect *temps-réel* qui indique que le comportement du système est contraint par rapport au temps. Par conséquent, l'effet d'un événement dépend de l'instant de son occurrence qui est contrôlée à l'aide soit des temporisateurs soit des contraintes d'horloges. Les pouvoirs expressifs des temporisateurs et des horloges sont identiques mais la modélisation par des horloges est souvent plus compacte que celle basée sur des temporisateurs. Les aspects *réactif* et *temps-réel* ne peuvent être dissociés l'un de l'autre. D'une part, une action (ou une réaction) du système est contrainte par le temps. D'autre part, l'écoulement du temps, lui seul, peut déclencher certaines réactions du système.

Dans cette thèse, nous nous limitons à la famille des systèmes réactifs à états. Pour cette famille de systèmes, le comportement futur ne dépend que de l'état courant. Lorsqu'un système est modélisé par un STE par exemple, son état est donné par l'état de contrôle courant appartenant à son STE. Dans notre approche, nous n'avons pas de tels états de contrôle car la spécification partielle du système n'est pas donnée sous forme d'un modèle à états tel que les STEs . Par conséquent, nous allons définir une caractérisation de l'état du système basée sur les valeurs des variables discrètes et des horloges que nous associons au système réactif temps-réel. En outre, nous supposons que le comportement du système peut être décrit comme un processus séquentiel. Cette limitation est imposée par le modèle cible de la spécification globale du système qui est sous forme d'un automate temporisé sécuritaire. Ainsi le système ne permet qu'un seul événement (ou action) à la fois.

Dans un premier temps, nous définissons un ensemble d'horloges associées au système. Ensuite, nous introduisons les variables discrètes du système ainsi que leurs éléments de base comme nous l'avons fait pour les horloges dans la section 3.2. Ces éléments de base regroupent les valuations de variables ainsi les syntaxes respectives des contraintes de variables et des affectations de variables. Par la suite, nous définissons les événements observables par le système et la structure de l'état du système.

5.1.1 Les horloges du système

Les horloges du système sont des variables globales continues à valeurs réelles positives. Nous notons H l'ensemble des horloges du système. Nous adoptons pour les horloges du système les mêmes syntaxe et sémantique que celles définies dans les automates temporisés au chapitre 3. Nous rappelons que $\Theta(H)$ désigne l'ensemble des valuations d'horloge, $\Phi(H)$ représente l'ensemble des contraintes d'horloges sur H respectant la syntaxe exprimée par l'équation (3.1). De même $\Phi^+(H)$ désigne l'ensemble des contraintes positives d'horloges sur H respectant la syntaxe de l'équation (3.2).

5.1.2 Les variables discrètes du système

Nous utilisons des variables discrètes pour décrire les propriétés non temporelles de l'état du système. Nous nous limitons aux systèmes réactifs dont les domaines des variables sont discrets et finis. Ainsi, le domaine des valeurs d'une variable discrète est isomorphe à un intervalle d'entier. Nous désignons par V l'ensemble des variables discrètes du système et notons $dom(v)$ le domaine d'une variable v appartenant à V.

Exemple 5.1. *Quelques formes possibles pour le domaine d'une variable*
$V = \{v_1, v_2, v_3, v_4\}$
$dom(v_1) = \{ON, OFF\}$
$dom(v_2) = \{1, 3, 7, 9\}$
$dom(v_3) = [1, .., 10]$
$dom(v_4) = \{1, 2, ON, OFF\}$
Le domaine de la variable v_1 est un ensemble de constantes. Une variables peut avoir un domaine qui ne comporte que des valeurs entières ; c'est le cas des domaines de v_2 et v_3. De plus, le domaine de v_3 est un intervalle. La possibilité d'avoir un domaine de variable composé d'une combinaison d'entiers et de constantes, est aussi permise comme le montre le cas de la variable v_4.

Dans le reste de cette sous-section nous définissons les valuations, les contraintes et les affectations de variables que nous utilisons dans la description de la spécification du système.

Valuation de variables

Une valuation (ou configuration) de variables est une application qui associe une valeur à chaque variable appartenant à V. Nous désignons par $\Omega(V)$ l'ensemble des valuations de variables

de V. Une valuation de variables est souvent notée ω. Pour chaque variable v appartenant à V, $\omega(v)$ appartient au domaine $dom(v)$ et désigne la valeur de la variable v par la valuation ω.

Étant fini, nous supposons que l'ensemble des variables V compte p variables qui sont v_1, v_2, $\ldots$, v_p. L'ensemble des valuations de variables $\Omega(V)$ est isomorphe au produit cartésien des domaines des variables $dom(v_1) \times \cdots \times dom(v_p)$.

Exemple 5.2. *Basé sur l'ensemble V défini dans l'exemple 5.1, un exemple de valuation de variables qui appartient à $\Omega(V)$ est le vecteur $\omega = (OFF, 7, 1, ON)$. Dans ce cas, nous avons $\omega(v_1) = OFF$, $\omega(v_2) = 7$, $\omega(v_3) = 1$ et $\omega(v_4) = ON$.*

Contraintes de variables

Les contraintes de variables sont des conditions sur leurs valeurs. Chaque contrainte est une expression logique selon la syntaxe suivante :

$$\psi ::= v\#c \mid \psi \wedge \psi \mid \psi \vee \psi \mid \neg\psi \mid True \mid False \tag{5.1}$$

où v est une variable appartenant à V, c est une constante appartenant à $dom(v)$ et $\#$ est une relation binaire de $dom(v) \times dom(v)$. Par exemple, si $dom(v)$ est muni d'une relation d'ordre total, la relation $\#$ peut être l'une de $\{\leq, <, =, \geq, >\}$. $\Psi(V)$ est l'ensemble des contraintes de variables respectant la syntaxe de l'équation (5.1).

Nous notons $\psi(\omega)$ la valeur booléenne représentant la valeur de vérité de ψ en ω. Si $\psi(\omega)$ est vrai, nous dirons que la valuation de variables ω satisfait la contrainte ψ. Nous notons $[\psi]$ l'ensemble des valuations de variables qui satisfont la contrainte ψ.

Exemple 5.3. *Dans cet exemple nous supposons que V est l'ensemble de variables définies dans l'exemple 5.1. Soient $\omega = (OFF, 7, 1, ON)$ une valuation de variables et $\psi_1 \stackrel{def}{=} (v_1 = OFF \wedge v_3 \geq 5)$ et $\psi_2 \stackrel{def}{=} (v_1 = ON \vee v_3 = 5)$ deux contraintes de variables appartenant à $\Psi(V)$. Dans ce cas, la valeur $\psi_1(\omega)$ est vraie alors que celle de $\psi_2(\omega)$ est fausse.*

Affectation de variables

Les événements observés entraînent un changement d'état du système. Les variables discrètes sont modifiées pour marquer ce changement en utilisant une affectation de variables qui est définie par un ensemble d'instructions. Nous ne faisons aucune restriction sur les instructions permises

pour définir une affectation sauf qu'une variable ne peut être modifiée plus d'une fois par dans la même affectation. Une affectation de variables peut être vue comme une fonction qui associe à une valuation de variable une nouvelle valuation calculée selon les instruction de l'affectation.

Nous notons δ une affectation de variables et $\Delta(V)$ l'ensemble des affectations de variables de V. L'application d'une affectation de variables δ à une valuation de variables ω donne lieu à une nouvelle valuation de variables que nous notons $\omega[\delta]$. Cette notation a été adoptée par similitude à celle des assignations d'horloges (section 3.2.3). La valeur affectée par la valuation $\omega[\delta]$ à chaque variable est donnée en fonction de l'instruction de δ modifiant cette variable. Les valuations ω et $\omega[\delta]$ associent la même valeur à une variable si aucune instruction de δ ne la modifie.

Exemple 5.4. *Nous supposons dans cette exemple que l'ensemble des variables V est celui défini dans l'exemple 5.1. Considérons l'affectation $\delta = \{v_1 := ON, v_3 := v_3 + 1\}$ et la valuation de variables $\omega = (OFF, 3, 1, OFF)$. Dans ce cas $\omega[\delta]$ est la valuation de variables définie par :*

$$\omega[\delta] : \begin{cases} \omega[\delta](v_1) = ON \\ \omega[\delta](v_3) = \omega(v_3) + 1 = 7 + 1 = 8 \\ \omega[\delta](v) = \omega(v) & \text{pour } v \in V \backslash \{v_1, v_3\} \end{cases}$$

Par contre, si nous considérons la valuation de variables $\omega' = (OFF, 3, 10, OFF)$, $\omega'[\delta]$ n'est pas définie car $\omega[\delta](v_3) = 10 + 1$ n'appartient pas au domaine de v_3 si la relation $+$ représente la somme d'entiers.

Étant données ψ une contrainte appartenant à $\Psi(V)$ et δ une affectation de variables, $\psi[\delta]$ est la contrainte satisfaite par toutes les valuations de variables $\omega[\delta]$ telles que ω satisfait ψ. La contrainte $\psi[\delta]$ existe car les domaines des variables sont finis et discrets.

L'ensemble des contraintes de variables $\Psi(V)$ est isomorphe à l'ensemble des parties de $\Omega(V)$. L'isomorphisme reliant $\Psi(V)$ et $\Omega(V)$ permet de confondre $[\psi]$ et ψ en utilisant les règles suivantes :

$$[\psi \wedge \psi'] = [\psi] \cap [\psi']$$

$$[\psi \vee \psi'] = [\psi] \cup [\psi']$$

$$[\psi \wedge \neg\psi'] = [\psi] - [\psi']$$

5.1.3 Les événements

Les événements observables sont représentés par des étiquettes et modélisent les interactions du système avec son environnement. Nous distinguons deux types d'événements : ceux que le système contrôle et ceux qu'il ne contrôle pas. Les événements contrôlables sont par exemple les envois de messages. Dans ce cas, c'est le système qui décide (selon sa spécification) d'envoyer ou pas un message. Lorsque le système a le choix entre deux événements contrôlables, il y a un non-déterminisme. Contrairement aux événements contrôlables, le système subit l'exécution des événements non contrôlables et qui sont imposés par l'environnement. La réception des messages fait partie des événements non contrôlables par le système. Nous notons Ev l'ensemble des événements du système.

5.1.4 Notations globales

Nous avons associé au système trois ensembles H, V et Ev. H représente un ensemble d'horloges. V désigne un ensemble de variables discrètes. Ev est un ensemble d'événements représentées par des étiquettes observables comme celles d'un STE. Le triplet (H, V, Ev) regroupe de l'information statique sur le domaine d'application du système.

Dans la suite de cette thèse, nous adoptons les notations décrites à la table 5.1. Les éléments φ, ψ, λ ... de cette table peuvent être utilisés avec un indice, par exemple, φ'_r indique que cette contrainte d'horloges est liée à l'entité r.

Notation	**Description**
$\theta \in \Theta(H)$	valuation d'horloges
$\omega \in \Omega(V)$	valuation de variables
$\varphi, \varphi', \varphi'' \in \Phi(H)$	contraintes d'horloges
$\psi \in \Psi(V)$	contrainte de variables
$\lambda \subset H$	assignation d'horloges
$\delta \in \Delta(V)$	affectation de variables

TAB. 5.1 – Table récapitulative des notations

5.1.5 Description de l'état d'un système

L'état du système est un couple (ω, θ) où ω est une valuation de variables appartenant à $\Omega(V)$ et où θ représente une valuation d'horloges à appartenant à $\Theta(H)$. Afin de soulager les notations manipulant un état $e = (\omega, \theta)$, nous adoptons les écritures suivantes :

- $\psi(e) = \psi(\omega)$
- $\varphi(e) = \varphi(\omega)$
- $e[\delta] = \omega[\delta]$
- $e[\lambda] = \theta[\lambda]$
- $e + d = (\omega, \theta + d)$

5.2 Description du comportement élémentaire

Le comportements du système est décrit par un ensemble de scénarios. Dans cette sous section nous donnons la forme canonique de la représentation des scénarios en fonction d'un ensemble *actions-règles* [SDL01a]. Nous aborderons au chapitre suivant comment se fait le passage entre la forme de saisie des scénarios et leurs formes canoniques. Pour le moment nous supposons que la spécification du système est décrite sous forme d'un ensemble d'actions-règles.

Une action-règle représente l'entité élémentaire de description du comportement du système et comporte toute l'information concernant l'exécution d'une action et l'évolution possible de l'état du système avant et après cette exécution.

Dans la suite de la section courante, nous donnons la syntaxe des actions-règles ainsi que leur sémantique opérationnelle. Ensuite, à partir d'un ensemble d'actions-règles, nous définissons un ATS qui lui est équivalent conformément à sa sémantique opérationnelle. Cet ATS représente une spécification qui intègre toutes les actions-règles de cet ensemble.

5.2.1 Définition syntaxique des actions-règles

Définition 5.1. $r = (\psi_r, \varphi_r, l_r, \varphi'_r, \delta_r, \lambda_r, \varphi''_r)$ *est une action-règle ssi :*

- ψ_r *: une contrainte de variables appartenant à* $\Psi(V)$ *et représentant une pré-condition pour l'exécution de l'action-règle* r.
- φ_r *: une contrainte positive d'horloges qui appartient à* $\Phi^+(H)$. φ_r *représente l'invariant ou la condition d'activité sur l'état du système lorsque cet état satisfait* ψ_r.

- l_r : *une étiquette observable utilisée pour la synchronisation.* l_r *appartenant à l'ensemble des événements* Ev.
- φ'_r : *une contrainte positive d'horloges qui représente la garde temporelle de l'action-règle* .
- δ_r : *une affectation de variables de* V *décrivant la mise à jour des variables après l'exécution de l'action-règle* r.
- λ_r : *une assignation d'horloges qui indique les horloges à initialiser après l'exécution de l'action-règle.*
- φ''_r : *une contrainte positive d'horloges représentant une condition d'activité sur l'état du système après l'exécution de l'action-règle* r. *Ainsi* φ''_r *représente l'invariant de l'état du système lorsque cet état satisfait la contrainte* $\psi_r[\delta_r]$ *c'est à dire après l'exécution de l'action-règle* r.

Une action-règle est une forme compacte qui décrit deux aspects du comportements :

(I) un aspect relié à l'écoulement du temps lorsque le système est passif soit avant l'exécution de l'action-règle soit après l'exécution de l'action-règle.

(II) et un aspect décrivant l'activité de l'action-règle qui se manifeste par l'événement de l'action-règle.

Une action-règle r signifie que le système a le potentiel de l'exécuter dans un état e où le prédicat $\psi_r(e) \wedge \varphi_r(e)$ est vrai. Cette condition rappelle l'invariant d'une place dans un ATS. L'action-règle r ne peut être effectivement exécutée que si l'état e du système satisfait la garde temporelle ϕ et que l'état du système après l'exécution satisfait l'invariant $\varphi''_r[\lambda_r]$. Après l'exécution de r, l'état du système est donné par l'expression $e' = (e[\delta_r], e[\lambda_r])$. De plus, à partir de tout état e tel que $\psi_r(e) \wedge \varphi_r(e)$ est vrai, l'action-règle r permet l'écoulement du temps tant que l'état du système satisfait la contrainte φ_r.

Exemple 5.5. *Considérons un système défini par la modélisation suivante :*

- $V = \{v_1, v_2\}$. *Les domaines des variables* v_1 *et* v_2 *sont données respectivement par* $dom(v_1) = \{0, 1, 2\}$ *et* $dom(v_2) = \{-2, 1, 0, 1, 2\}$.
- $H = \{h\}$
- $Ev = \{in, out\}$

Soit $\hat{r} = (\psi_{\hat{r}}, \varphi_{\hat{r}}, l_{\hat{r}}, \varphi'_{\hat{r}}, \delta_{\hat{r}}, \lambda_{\hat{r}}, \varphi''_{\hat{r}})$ *l'action-règle décrite par :*

- $\psi_{\hat{r}}$: $v_1 = 0$

- $\varphi_{\hat{r}}$: $h \leq 30$
- $l_{\hat{r}}$: *out*
- $\varphi'_{\hat{r}}$: $h \leq 20$
- $\delta_{\hat{r}}$: $v_1 := v_1 + 1$
- $\lambda_{\hat{r}}$: $\{h\}$
- $\varphi''_{\hat{r}}$: $h \leq 60$

*Nous allons décrire concrètement le comportement de l'action-règle $\hat{r}$. L'exécution de l'action règle $\hat{r}$ ne peut se faire qu'à partir des états du système qui satisfont les contraintes $v_1 = 0$ et $h \leq 30$. Vu que les valeurs des horloges sont des nombres réels, le nombre d'états du système qui satisfont les contrainte l'action-règle $\hat{r}$ est infini. Le point (**I**) précédent exprime le fait qu'avant l'exécution de $\hat{r}$ le temps peut s'écouler tant que $h \leq 30$ est satisfaite et qu'après l'exécution de $\hat{r}$ le temps peut s'écouler tant que $h \leq 60$ est satisfaite. Le point (**II**) précédent se traduit par l'exécution de l'action-règle $\hat{r}$ qui correspond à l'observation de l'événement out. De plus, cette exécution n'est permise que si l'état du système satisfait la garde temporelle $h \leq 20$. L'exécution de l'action-règle $\hat{r}$ se traduit par un changement dans l'état du système qui consiste à incrémenter la valeur de la variable v_1 et à remettre à zéro l'horloge h.*

Si nous remplaçons $\varphi'_{\hat{r}} = h \leq 20$ par $\varphi'_{\hat{r}} = h \geq 80$ nous obtenons une action-règle qui ne va jamais s'exécuter. Nous disons qu'elle est inconsistante.

Comme le montre l'exemple précédent, une action-règle peut être inconsistante si l'une des contraintes de variables ou d'horloges est insatisfaisable. Nous définissons une action-règle consistante par :

Définition 5.2. *Une action-règle $r = (\psi_r, \varphi_r, l_r, \varphi'_r, \delta_r, \lambda_r, \varphi''_r)$ est consistante ssi :*

1. $[\psi_r] \neq \emptyset$
2. $[\varphi_r] \cap [\varphi'_r] \neq \emptyset$
3. $[(\varphi_r \wedge \varphi'_r)[\lambda_r]] \subset [\varphi''_r]$

Le point (1.) indique qu'il existe une valuation de variables pour laquelle l'action-règle r peut être exécutée. La condition (2.) exprime que l'invariant φ_r de l'action-règle peut être satisfait en même temps que la garde φ'_r. La condition (2.) indique aussi le cas où les des contraintes φ_r ou φ'_r sont satisfaisables. Par contre, le point (3.) signifie que l'invariant temporelle φ''_r doit être satisfait après l'exécution de l'action-règle r.

Dans la sous-section suivante, nous formalisons cette sémantique et nous l'associons aux actions-règles.

5.2.2 Sémantique des actions-règles

Un ensemble d'actions-règles fournit une description partielle du comportement du système. Nous qualifions cette description de partielle car un ensemble d'actions-règles ne décrit pas forcément tous les comportements du système. Chaque action-règle décrit un événement et contraint l'écoulement du temps de manière indépendante des autres actions-règles. Mais de manière globale, un ensemble actions-règles représente une description implicite d'un système de transitions étiquetées dont les transitions sont déduites des actions-règles. Le système de transitions étiquetées d'un ensemble actions-règles (STE-EAR) reflète sa sémantique globale.

Définition 5.3. *Nous notons Σ_R le STE de l'ensemble d'actions-règles de R (en abrégé STE-EAR de R). L'ensemble des états de Σ_R est un sous-ensemble de $\Omega(V) \times \Theta(H)$. La relation de transition $\rightarrow$ de Σ_R est définie à partir des actions-règles de R. Pour chaque $r \in R$, l'action-règle r représente deux types de transitions :*

- *des transitions instantanées de la forme $e \xrightarrow{l_r} e'$ telles que e appartient à $\Omega(V) \times \Theta(H)$ et $e' = (e[\delta_r], e[\lambda_r])$ et à condition que $\psi_r(e) \wedge \varphi(e) \wedge \varphi'(e[\lambda_r])$ soit vrai.*
- *et des transitions d'écoulement du temps de la forme $e \xrightarrow{d} e + d$ telles que e appartient à $\Omega(V) \times \Theta(H)$ et où d est un réel positif tel que l'un des deux prédicats suivants soit vrai :*
  - $\psi_r(e) \wedge (\forall 0 \leq d' \leq d \,.\, \varphi_r(e + d'))$
  - $\psi_r[\delta_r](e) \wedge (\forall 0 \leq d' \leq d \,.\, \varphi''_r(e + d'))$

De plus, la relation de transition de Σ_R vérifie la propriété d'additivité des transitions d'écoulement du temps. Pour tous d et d' deux nombres réels positifs, si Σ_R comporte les transition $e \xrightarrow{d} e'$ et $e' \xrightarrow{d'} e''$ alors $e \xrightarrow{d+d'} e''$ est aussi une transition de Σ_R.

Nous remarquons que la sémantique d'un ensemble d'actions-règles est définie par son STE-EAR. La sémantique d'un ATS est aussi définie par un STE qui a la même structure que les STE-EARs. L'idée est de trouver un ATS dont le STE est exactement un STE-EAR.

5.2.3 Automate temporisé sécuritaire d'un ensemble d'action-règle

La structure du STE-EAR est calquée sur celle du STE d'un ATS. Le STE-EAR représente un modèle global intégrant ses actions-règles mais qui présente l'inconvénient d'avoir une représentation infinie. La construction d'un ATS modélisant un ensemble d'actions-règles permet d'éliminer

cet inconvénient en fournissant une représentation finie du STE-EAR.

Théorème 5.1. *Soit R un ensemble d'actions-règles consistantes. Il existe A_R un ATS dont le STE est exactement le STE-EAR Σ_R de R.*

A_R est appelé automate temporisé sécuritaire de l'ensemble d'actions-règles de R, en abrégé ATS-EAR de R.

Démonstration. Par construction
Nous définissons la fonction $Time$ de $\Omega(V)$ vers l'ensemble des parties de $\Phi^+(H)$ par :

$$Time(\omega) = \{\varphi_r \mid \exists r \in R \,.\, \omega \in [\psi_r]\} \cup \{\varphi''_r \mid \exists r \in R \,.\, \omega \in [\psi_r[\delta_r]]\} \tag{5.2}$$

Nous construisons l'automate temporisé sécuritaire $A_R = (L_{A_R}, L^o_{A_R}, M_{A_R}, T_{A_R}, Inv_{A_R})$ par :

- $L_{A_R} = \{\omega \in \Omega(V) \mid \exists r \in R \,.\, \omega \in [\psi_r] \cup [\psi_r[\delta_r]]\}$
- $L^o_{A_R} = L_{A_R}$
- $M_{A_R} = Ev$
- $T_{A_R} = \{\omega, l_r, \varphi_r \wedge \varphi'_r, \lambda_r, \omega[\delta_r]\}$
- $Inv_{A_R}(\omega) = \bigvee_{\varphi \in Time(\omega)} \varphi$

Par construction, le STE de l'automate A_R est exactement Σ_R □

L'association d'un automate temporisé à un STE qui modélise l'écoulement du temps n'est pas toujours possible car l'ensemble des automates temporisés n'est pas fermé pour le complément. Ce résultat à été prouvé par Alur et Dill [AD94]. Nous nous inspirons de leur preuve pour donner un contre-exemple qui prouve qu'il n'est pas toujours possible d'associer un automate temporisé à un STE.

Contre-exemple : Considérons L'ATS A représenté à la figure 5.1 et défini par :

- $L_A = \{s\}$
- $L_A{}^o = \{s\}$
- $M_A = \{l\}$
- $H_A = \{h\}$
- $T_A = \{(s, l, True, \emptyset, s)\} \subset L_A \times M_A \times \Phi(H_A) \times 2^{H_A} \times L_A$
- $Inv_A(s) = True$

Posons $\Sigma_A = (Q, Q_o, \rightarrow)$ le STE associé à l'automate temporisé A et considérons la relation de transition suivante :

$$\rightarrow' = \rightarrow \setminus \{(s,d) \xrightarrow{l} (s,d) \quad | \quad d \in \mathbb{N}^*\}$$

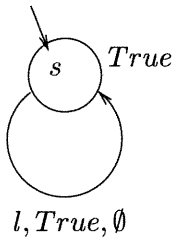


FIG. 5.1 – Automate temporisé sécuritaire

Où $\mathbb{N}^*$ est l'ensemble des entier naturels non nuls.

Il n'existe pas d'automate temporisé dont le système de transition est $(Q, Q_o, \rightarrow')$ car si un tel automate temporisé existait, il devrait avoir un nombre infini d'horloges et de transitions.

5.3 Réduction de l'ATS-EAR

Selon la construction l'ATS-EAR dans le théorème 5.1, chaque place de l'ATS-EAR est caractérisée par une valuation de variables. Par conséquent, l'ATS-EAR présente le risque d'une explosion combinatoire du nombre de places. Nous réduisons l'ATS-EAR par une relation d'équivalence. Le principe de cette réduction consiste à regrouper les places de l'ATS-EAR en classes d'équivalence de telle sorte que toutes les places d'une classe ont les mêmes actions-règles sortantes et les mêmes actions-règles entrantes. Ce principe est illustré par la figure 5.2. La classe $[\psi]$ regroupe toutes les places ω de l'ATS-EAR qui ont $r_1, r_2, .., r_k$ comme transitions sortantes et $r'_1, r'_2, .., r'_{k'}$ comme transitions entrantes.

Etant donné un ensemble d'actions-règles R, nous réduisons le problème de la réduction de l'ATS-EAR de R en un problème de minimisation d'un STE $\Sigma'_R = (Q, Q_o, \rightarrow)$ défini par :

- $Q = \{\omega \in \Omega(V) \mid \exists \psi \in \Psi_R \cup \Psi'_R \,.\, \omega \in [\psi]\}$
- $\rightarrow = \{(\omega, r, \omega[\delta_r]) \mid \exists\, r \in R \,.\, \omega \in [\psi_r]\}$
- Q_o est l'ensemble des états initiaux. Nous posons $Q_o = Q$.

Le choix de $Q_o = Q$ se justifie par le fait que la spécification du système est en cours de conception et risque d'être incomplète. Par conséquent l'état initial peut être encore inconnu. Ce choix

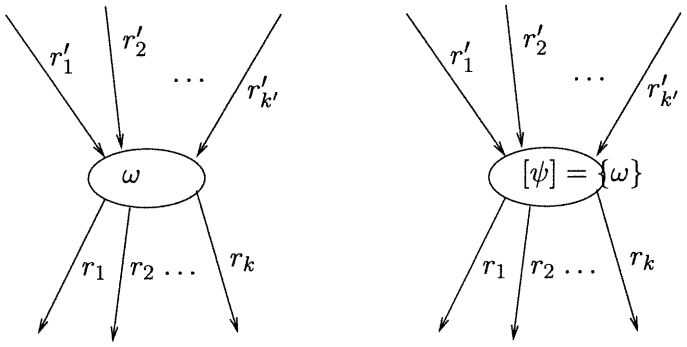


FIG. 5.2 – Principe de la minimisation d'un ATS-EAR

est supporté par le principe de l'engagement minimal.

Nous définissons une relation d'équivalence $\sim$ entre les états de Σ'_R :

Définition 5.4. *Soient ω_1 et ω_2 deux états de Σ'_R. ω_1 et ω_2 sont dits équivalents et notés $\omega_1 \sim \omega_2$ ssi :*

$(1a)$ $\forall r \in R$, $(\exists \omega'_1 \in Q \,.\, \omega_1 \xrightarrow{r} \omega'_1)$ *implique* $(\exists \omega'_2 \in Q \,.\, \omega_2 \xrightarrow{r} \omega'_2$ *telle que* $\omega'_1 \sim \omega'_2)$.

$(1b)$ $\forall r \in R$, $(\exists \omega'_2 \in Q \,.\, \omega_2 \xrightarrow{r} \omega'_2)$ *implique* $(\exists \omega'_1 \in Q \,.\, \omega_1 \xrightarrow{r} \omega'_1$ *telle que* $\omega'_1 \sim \omega'_2)$.

$(2a)$ $\forall r \in R$, $(\exists \omega'_1 \in Q \,.\, \omega'_1 \xrightarrow{r} \omega_1)$ *implique* $(\exists \omega'_2 \in Q \,.\, \omega'_2 \xrightarrow{r} \omega_2)$

$(2b)$ $\forall r \in R$, $(\exists \omega'_2 \in Q \,.\, \omega'_2 \xrightarrow{r} \omega_2)$ *implique* $(\exists \omega'_1 \in Q \,.\, \omega'_1 \xrightarrow{r} \omega_1)$

Les conditions $(1a)$ et $(1b)$ expriment que la relation d'équivalence $\sim$ énoncée par la définition 5.4 est une bisimulation. Les points $(2a)$ et $(2b)$ forment la condition de surjectivité des classes d'équivalence de $\sim$ et elle exprime la surjectivité de l'application transition sur les classes d'équivalence. En d'autres termes : s'il existe une transition de Σ'_R étiquetée par r dont l'état d'arrivée appartient à une classe d'équivalence, alors tous les états de cette classe d'équivalence sont des états d'arrivée d'une transition étiqueté par r. De manière intuitive, la condition de surjectivité permet de regrouper les places du ATS-EAR qui ont le même invariant temporel. Par contre, les conditions $(1a)$ et $(1b)$ permettent de regrouper ensemble les places du ATS-EAR à partir desquelles les mêmes actions-règles peuvent être exécutées.

La minimisation de Σ'_R consiste à calculer les classes d'équivalence de la relation $\sim$ donnée par la définition 5.4. En général, le problème de la minimisation d'un STE peut être réduit au calcul de la plus grande bisimulation raffinant une partition initiale judicieusement choisie [HU79, BFH+92]. Puisque la relation $\sim$ est une bisimulation, la réduction de Σ'_R par rapport à la relation $\sim$ revient à calculer la plus grande relation de bisimulation raffinant une partition initiale vérifiant certaines conditions que nous aborderons plus loin.

Dans cette section, nous décrivons un algorithme qui calcule la plus grande bisimulation raffinant une partition initiale. Ensuite, nous considérons le problème de la détermination de la partition initiale dans le but de minimiser Σ'_R par rapport à la relation d'équivalence $\sim$. Enfin nous utilisons les résultats des deux étapes précédentes pour traiter la réduction de l'ATS-EAR de R.

5.3.1 La plus grande bisimulation de Σ'_R raffinant une partition initiale

Le calcul de la plus grande relation de bisimulation contenant une partition initiale a été étudié dans plusieurs travaux de la théorie des automates et des langages réguliers [HU79, KS83, PT87, Fer90, BFH90, ACH+92, KL94]. Les algorithmes de calcul de la plus grande relation de bisimulation raffinant une partition initiale se basent sur le même principe qui consiste à raffiner progressivement la partition initiale jusqu'à l'obtention d'une partition stable qui définit une relation de bisimulation.

L'algorithme BFH désigne dans le reste du chapitre, l'algorithme de minimisation décrit dans [BFH90, BFH+92]. L'algorithme BFH est un algorithme générique de minimisation permettant de calculer la plus grande relation de bisimulation, par rapport à une partition initiale, dans le cas d'un système de transitions non étiquetées. La généricité de l'algorithme BFH se manifeste par le fait qu'il n'est pas nécessaire d'avoir une description explicite du système de transitions. Toute description implicite, par exemple sous forme d'un programme, est utilisable par l'algorithme sans avoir à développer en mémoire tous l'espace d'états. Cette caractéristique est utile dans le cas de la minimisation de Σ'_R qui est en fait implicitement décrit par les actions-règles de R.

Nous décrivons brièvement l'algorithme BFH avant de le généraliser et de l'adapter pour l'appliquer dans la minimisation de Σ'_R.

Description de l'algorithme de BFH

L'algorithme BFH permet de minimiser un système de transitions non étiquetées par rapport à une partition initiale de l'ensemble des états. Nous désignons par système de transitions non étiquetées, un système de transitions étiquetées dont l'ensemble des étiquettes est réduit à une seule étiquette. Soit $(E, E_o, \rightarrow)$ un système de transitions non étiquetées. Les transitions dans ce système sont notées $q \rightarrow q'$ telles que q et q' sont des états appartenant à E. Nous désignons par ρ une partition de E. Les éléments de ρ sont appelés des classes. Pour tout état q appartenant à E, nous notons $post_\rho(q)$ l'ensemble des classes de ρ immédiatement accessibles à partir de q :

$$post_\rho(q) = \{C \in \rho \quad | \quad \exists q' \in C \quad et \quad q \rightarrow q'\}$$

Nous étendons l'opérateur $post_\rho$ à l'ensemble des sous-parties de E. Ainsi pour $C \subset E$ nous définissons :

$$post_\rho(C) = \underset{q \in C}{\cup} post_\rho(q)$$

Une classe $C \in \rho$ est dite stable par rapport à la partition ρ de E si pour chaque classe $C' \in \rho$ et pour chaque transition $(q_1 \rightarrow q_1')$ telle que $q_1 \in C$ et $q_1' \in C'$ alors pour tout $q_2 \in C$ il existe $q_2' \in C'$ tels $(q_2 \rightarrow q_2')$ est une transition. De manière plus formelle, une classe C appartenant à ρ est dite stable ssi :

$$\forall C' \in \rho \quad \forall (q_1, q_1') \in C \times C' \quad (q_1 \rightarrow q_1' \quad implique \quad (\forall q_2 \in C \; \exists q_2' \in C' \quad . \quad q_2 \rightarrow q_2'))$$

L'algorithme BFH se base sur l'opérateur $split(C, \rho)$ qui permet de partitionner la classe C en classes stables par rapport à ρ. L'opérateur $split$ est défini par :

$$\begin{aligned} split(C, \rho) \quad = \quad & \{C_1, \ldots, C_k \mid (\forall i, j . C_i \cap C_j = \emptyset) \; et \; (C_1 \cup \cdots \cup C_k = C) \\ & et \; (\forall i \; . \; C_i \; stable) \; et \; (\forall i, j \; . \; C_i \cup C_j \; non \; stable)\} \end{aligned} \tag{5.3}$$

L'algorithme de BFH (figure 5.3) consiste à raffiner progressivement une partition ρ initialisée à ρ_o comme partition initiale. À chaque étape, l'ensemble Acc représente l'ensemble des classes contenant au moins un état accessible à partir de l'état initial et l'ensemble Stb regroupe les classes stables par rapport à ρ. La procédure BFH s'arrête lorsque toutes les classes de la partition courante ρ sont stables.

procédure BFH

Input : ρ_o partition initiale,

Output : ρ partition stable

Local : Stb ensemble des classes stables, Acc ensemble des classes accessibles

$\rho := \rho_o$; $Acc := \{C \in \rho | E_o \cap C \neq \emptyset\}$; $Stb := \emptyset$;

tant que $Acc \neq Stb$ **faire**

choisir $C \in Acc - Stb$

$D := split(C, \rho)$

si $D = \{C\}$ **alors**

$Stb := Stb \cup \{C\}$; $Acc := Acc \cup post_\rho(C)$

sinon

$Acc := (Acc - C) \cup \{C' \in D | C' \cap E_o \neq \emptyset\}$;

$Stb := Stb - \{C' \in Stb | C \in post_\rho(C')\}$;

$\rho := (\rho - \{C\}) \cup D$;

FIG. 5.3 – Algorithme BFH [BFH90]

Nous notons $\bar{\rho}$ la partition résultante de la procédure de minimisation. Toutes les classes de $\bar{\rho}$ sont stables. La relation d'équivalence $B_{\bar{\rho}}$ est définie par :

$$B_{\bar{\rho}} = \{(q, q') \in E \times E | \exists C \in \bar{\rho} \,.\, q \in C \; et \; q' \in C\}$$

est une bisimulation entre le système de transitions $(E, E_o, \rightarrow)$ et lui même. $B_{\bar{\rho}}$ est la plus grande bisimulation incluse dans ρ_o [BFH+92].

Le système de transitions non étiquetées $(E, E_o, \rightarrow)$ est réduit en un système de transitions non étiquetées quotient noté par $(\bar{\rho}, \{C \in \bar{\rho} | C \cap E_o \neq \emptyset\}, \rightarrow_{\bar{\rho}})$ et qui lui est bisimilaire. La relation de transition $\rightarrow_{\bar{\rho}}$ est définie par :

$$\forall C, C' \in \bar{\rho} \quad (C \rightarrow_{\bar{\rho}} C' \quad ssi \quad \exists (q, q') \in C \times C' \,.\, q \rightarrow q')$$

Généralisation de l'algorithme BFH au cas d'un STE quelconque et à Σ'_R en particulier

Nous adaptons et généralisons la procédure BFH (figure 5.3) pour l'utiliser dans la minimisation de Σ'_R. Nous effectuons deux adaptations :

- La première adaptation est une généralisation qui permet d'étendre l'algorithme BFH pour traiter tout système de transitions étiquetées et non seulement ceux dont l'alphabet est restreint à une seule étiquette. Pour cela, il suffit de généraliser la définition de la stabilité des classes par rapport à une partition.
- La deuxième adaptation est une simplification de l'algorithme BFH qui évite l'analyse d'accessibilité. Dans le cas de Σ'_R tous les états sont supposés des états initiaux et donc ils sont accessibles. Par conséquent, nous éliminons l'ensemble Acc dans la procédure de minimisation et nous le remplaçons par ρ. Nous obtenons ainsi la procédure $\omega\text{-}minimisation$ que nous décrivons à la figure 5.4.

Nous étendons la définition de la stabilité des classes dans le cas d'un STE quelconque en particulier pour le STE $\Sigma'_R = (Q, Q_o, \rightarrow)$. Soit ρ une partition de Q. Nous savons que les éléments de ρ sont des classes comme nous l'avons déjà mentionné. Dans le cas de Σ'_R, une classe de ρ est un ensemble de valuations de variables. Par conséquent il peut être désigné par la contrainte de variables ψ qui définit cette classe de ρ. Une classe ψ appartenant à ρ est dite stable par rapport à ρ ssi :

$$\forall \psi' \in \rho \; \forall (\omega_1, \omega'_1) \in [\psi] \times [\psi'] \; \forall r \in R \,.\, (\omega_1 \xrightarrow{r} \omega'_1 \; implique \; (\forall \omega_2 \in [\psi] \; \exists \omega'_2 \in [\psi'] \,.\, \omega_2 \xrightarrow{r} \omega'_2))$$

procédure ω-*minimisation*

Input : ρ_o,

Output : ρ partition stable

Local : Stb ensemble des classes stables

$\rho := \rho_o$; $Stb := \emptyset$

tant que $(\rho \neq Stb)$ **faire**

choisir $\psi \in (\rho - Stb)$

$D := split(\psi, \rho)$

si $D = \{\psi\}$ **alors**

$Stb := Stb \cup \{\psi\}$

sinon

$Stb := Stb - \{\psi' \in Stb | \psi \in post_\rho(\psi')\}$;

$\rho := (\rho - \{\psi\}) \cup D$;

FIG. 5.4 – Algorithme de la procédure ω-*minimisation*

La procédure ω-*minimisation* utilise la fonction $split(\rho, \psi)$ qui consiste à répartir la classe $[\psi]$ en classes stables par rapport à ρ. L'expression de $split(\rho, \psi)$ est celle de l'équation (5.3). L'algorithme pour calculer $split(\rho, \psi)$, donné à la figure 5.5, utilise la fonction $pre_r(\psi)$ qui retourne l'ensemble des états de Q à partir desquels nous aboutissons à un état appartenant à $[\psi]$ en suivant une transition de Σ'_R étiquetée par $r \in R$:

$$pre_r(\psi) = \{\omega \in [\psi_r] \quad | \quad \omega[\delta_r] \in [\psi]\}$$

La procédure ω-*minimisation* calcule encore la plus grande relation de bisimulation raffinant la partition initiale [PT87, Fer90].

5.3.2 Détermination de la partition initiale

Notre objectif est de calculer les classes d'équivalence de la relation $\sim$ en utilisant la procédure ω-*minimisation*. Pour que la bisimulation calculée par la procédure ω-*minimisation* soit exac-

fonction split(ρ, ψ)
  $C = \{\psi\}$
  pour chaque $\psi' \in \rho$
    pour chaque $r \in R$
      $C' = \emptyset$
      pour chaque $\psi'' \in C$ **faire**
        $\hat{\psi} = pre_r(\psi') \cap [\psi'']$
        si ($\hat{\psi} = \emptyset$ ou $\hat{\psi} = \psi''$) **alors** $C' = C' \cup \{\psi''\}$
        sinon $C' = C' \cup \{[\hat{\psi}], ([\psi''] \backslash [\hat{\psi}])\}$
      $C = C'$
  retourner (C)

FIG. 5.5 – La fonction *split*

tement la relation d'équivalence $\sim$, il faut choisir une partition initiale qui contient la partition définie par les classes d'équivalence de $\sim$ qui forme en fait l'ensemble quotient de la relation d'équivalence $\sim$.

La partition définie par les classes d'équivalence de $\sim$ vérifie les conditions de la définition 5.4 : les conditions $(1a)$, $(1b)$ concernent la bisimulation et la condition (2) concerne la surjectivité. Par conséquent, La partition définie par les classes d'équivalence de $\sim$ est incluse dans la plus grande relation de bisimulation de Σ'_R et dans la plus grande partition vérifiant la condition de surjectivité. Nous allons calculer la plus grande partition vérifiant la condition de surjectivité que nous choisissons comme partition initiale à fournir à la procédure ω-*minimisation*. Dans la suite de cette section nous allons construire une partition puis montrer que cette partition est la plus grande partition vérifiant la condition de surjectivité.

Nous notons Ψ_R (respectivement Ψ'_R) l'ensemble des classes des valuations de variables de départ (respectivement d'arrivée) des actions-règles appartenant à R. Nous écrivons :

$$\Psi_R = \{\psi \in \Psi(V) \mid \exists r \in R \,.\, \psi = \psi_r\} \tag{5.4}$$

$$\Psi'_R = \{\psi \in \Psi(V) \mid \exists r \in R \,.\, \psi = \psi_r[\delta_r]\} \tag{5.5}$$

Soit P l'ensemble des sous-parties de Q. P est défini par :

$$P = \Psi'_R \cup \{Q \backslash (\underset{\psi \in \Psi'_R}{\cup} [\psi])\} \tag{5.6}$$

L'ensemble P, exprimé par l'équation (5.6), n'est pas une partition, mais la réunion de ses éléments couvre tous l'ensemble des états Q de Σ'_R. Nous désignons par ρ_o la plus grande partition de Q incluse dans P. La partition ρ_o est la plus grande partition de Q vérifiant la propriété suivante :

$$\forall \psi \in P \; \exists k > 0 \quad \exists \psi_1, \psi_2, ..., \psi_k \in \rho_o \mid [\psi] = [\psi_1] \cup [\psi_2] \cup ... \cup [\psi_k] \tag{5.7}$$

Pour calculer ρ_o, chaque élément de P est partitionné par rapport aux autres éléments de P. Nous construisons ρ_o en partitionnant chaque élément ψ appartenant à P en utilisant l'opérateur $part(\psi)$ défini par :

$$part(\psi) = \{\psi'' \in \Psi(V) \mid [\psi''] = ([\psi] \cap \bigcap_{\psi' \in P} D_{\psi'}) \; et \; D_{\psi'} \in \{[\psi'], Q - [\psi']\} \; et \; [\psi''] \neq \emptyset\} \tag{5.8}$$

La partition ρ_o est constituée par l'union des partitions des éléments de P :

$$\rho_o = \underset{\psi \in P}{\cup} part(\psi) \tag{5.9}$$

Nous remarquons que les $part(\psi)$ dans l'équation (5.9) peuvent avoir des éléments communs.

Proposition 5.1. *La partition ρ_o est la plus grande partition de Q qui satisfait la propriété de surjectivité :*

$$\forall \psi \in \rho_o \; \forall \omega_1 \in [\psi] \,.\, (\exists \omega'_1 \in Q \,.\, \omega'_1 \xrightarrow{r} \omega_1) \; implique \; (\forall \omega_2 \in [\psi] \,,\, \exists \omega'_2 \in Q \,.\, \omega'_2 \xrightarrow{r} \omega_2) \tag{5.10}$$

Démonstration. (i) ρ_o satisfait la propriété de surjectivité (5.10) par construction

(ii) Soit ρ une partition de Q qui satisfait la propriété (5.10). Nous allons montrer que ρ est incluse dans ρ_o c'est à dire :

$$\forall \psi \in \rho \; \exists \psi' \in \rho_o \,.\, [\psi] \subset [\psi']$$

Soit $\psi \in \rho$.

1er cas : Si pour tout r appartenant à R, $[\psi] \cap [\psi_r[\delta_r]] = \emptyset$. Dans ce cas, $[\psi] \subset Q - [\Psi']$ et $Q - [\Psi'] \in \rho_o$. Donc, ψ satisfait (ii)

2ème cas : Sinon, posons :

$X = \{\psi_r[\delta_r] \in P | [\psi_r[\delta_r]] \cap [\psi] \neq \emptyset\}$

et $Y = \{\psi_r[\delta_r] \in P | [\psi_r[\delta_r]] \cap [\psi] = \emptyset\}$

Nous remarquons d'après la propriété (5.10) que :

$$\forall \psi' \in X \,.\, [\psi] \subset [\psi'] \tag{5.11}$$

$$\forall \psi' \in Y \,.\, [\psi] \subset Q - [\psi'] \tag{5.12}$$

(5.11) implique $[\psi] \subset \bigcap_{\psi' \in X} [\psi']$

(5.12) implique $[\psi] \subset \bigcap_{\psi' \in Y} (Q - [\psi'])$

Nous remarquons aussi que $X \cup Y = P - \{Q - [\Psi']\}$ car $Q - [\Psi']$ est la classe dont les éléments ne reçoivent aucune transition dans Σ'_R.

Par contre, $[\psi] \subset [\Psi'] = Q - (Q - [\Psi'])$ ce qui implique que

$$[\psi] \subset \bigcap_{\psi' \in P - X} (Q - [\psi']) \tag{5.13}$$

En composant (5.11) et (5.13), nous obtenons :

$$[\psi] \subset (\bigcap_{\psi' \in X} [\psi'] \cap \bigcap_{\psi' \in P - X} (Q - [\psi'])) \in \rho_o \tag{5.14}$$

L'équation (5.14) est identique à la définition de ρ_o à partir de (5.9) et (5.8).

□

L'utilisation des formules (5.9) et (5.8) pour calculer ρ_o, aboutit à un nombre exponentiel d'intersections dont la majorité sont vides. L'algorithme de la procédure *partition* décrit à la figure 5.6 calcule plus efficacement la partition ρ_o. Il utilise la même technique que la fonction *split* de la figure 5.5.

En appliquant la procédure *ω-minimisation* sur le système de transition Σ'_R avec ρ_o comme partition initiale, nous obtenons la partition $\bar{\rho}$. Nous désignons par $B_{\bar{\rho}}$ la relation d'équivalence telle que $\bar{\rho}$ est son ensemble quotient. $B_{\bar{\rho}}$ est la plus grande bisimulation raffinant ρ_o [BFH+92].

procédure *partition*

Input : P : défini par l'équation (5.6),

Output : ρ_o : calculé selon l'équation (5.9),

$\rho_o = \emptyset$

tant que $(\exists \psi \in P \mid [\psi] \neq \emptyset)$ **faire**

  choisir $\psi \in P$

  $C := \{\psi\}$; $P := P - \{\psi\}$

  pour chaque $\psi' \neq \emptyset \in P$ **faire**

    $C' := \emptyset$

    pour chaque $\psi'' \in C$

      si $[\psi'] \cap [\psi''] = \emptyset$ ou $[\psi''] \subset [\psi']$

      alors $C' := C' \cup \{\psi''\}$

      sinon $C' := C' \cup \{[\psi''] - [\psi'], [\psi'] \cap [\psi'']\}$

      $\psi' := [\psi'] - [\psi'']$

    $C := C'$;

  $\rho_o := \rho_o \cup C$

FIG. 5.6 – Calcul de la partition ρ_o

Proposition 5.2. *La relation d'équivalence $B_{\bar{\rho}}$ est exactement la relation $\sim$*

Démonstration. $B_{\bar{\rho}}$ est la plus grande bisimulation raffinant ρ_0. D'après la proposition 5.1, nous déduisons que $B_{\bar{\rho}}$ est la plus grande bisimulation qui satisfait la condition de surjectivité. D'où $B_{\bar{\rho}}$ est égale à $\sim$.

□

$\bar{\rho}$ est l'ensemble quotient de la relation $\sim$. Nous allons utiliser la partition $\bar{\rho}$ pour la réduction, par rapport $\sim$, de l'ATS-EAR de R.

5.3.3 Réduction de l'ATS-EAR en un automate temporisée sécuritaire quotient

Les places de l'ATS-EAR sont des valuations de variables. Dans le but de réduire l'ATS-EAR en un ATS-EAR quotient, nous regroupons les places appartenant à la même classe d'équivalence de $\sim$ pour former une place de ATS-EAR quotient. Pour que ce regroupement soit possible, il faut que les places de l'ATS-EAR aient le même invariant. D'où le corollaire suivant :

Corollaire 5.1. *Les places de l'ATS-EAR de R qui appartiennent à la même classe d'équivalence de $\sim$ ont le même invariant.*

Démonstration. A_R représente l'ATS-EAR de R. Nous rappelons l'expression de l'invariant d'une place de A_R. L'expression de l'invariant d'une place ω dans A_R est définie dans la démonstration du théorème 5.1 par :

$$Inv_{A_R}(\omega) = \bigvee_{\varphi \in Time(\omega)} \varphi$$

tel que,

$$Time(\omega) = \{\varphi_r \mid \exists r \in R \,.\, \omega \in [\psi_r]\} \cup \{\varphi''_r \mid \exists r \in R \,.\, \omega \in [\psi_r[\delta_r]]\}$$

Quelque soit ω' qui appartient à la même classe que ω, nous déduisons du fait que $\sim$ est une bisimulation que :

$$\{\varphi_r \mid \exists r \in R \,.\, \omega \in [\psi_r]\} = \{\varphi_r \mid \exists r \in R \,.\, \omega' \in [\psi_r]\}$$

De même nous déduisons du fait que $\sim$ vérifie la condition de surjectivité que :

$$\{\varphi''_r \mid \exists r \in R \,.\, \omega \in [\psi_r[\delta_r]]\} = \{\varphi''_r \mid \exists r \in R \,.\, \omega' \in [\psi_r[\delta_r]]\}$$

Par conséquent,

$$Time(\omega) = Time(\omega')$$

D'où

$$Inv_{A_R}(\omega) = Inv_{A_R}(\omega')$$

Par conséquent, nous pouvons étendre l'application $Time$ aux classes d'équivalence de $\sim$ par :

$$\forall \psi \in \bar{\rho},\ \forall \omega \in [\psi]\ Time(\psi) = Time(\omega)$$

□

Nous définissons l'ATS-EAR quotient de A_R par rapport à $\sim$ comme étant l'automate temporisé sécuritaire $A_R/\sim = (L_{A_R/\sim}, L^o_{A_R/\sim}, M_{A_R/\sim}, T_{A_R/\sim}, Inv_{A_R/\sim})$ où :

- $L_{A_R/\sim} = \bar{\rho}$
- $L^o_{A_R/\sim} = \bar{\rho}$
- $M_{A_R/\sim} = Ev$
- $T_{A_R/\sim} = \{(\psi, l_r, \varphi_r \wedge \varphi'_r, \lambda_r, \psi') \mid$
  $\exists r \in R\ \exists \psi' \in \bar{\rho}\ .\ [\psi] \subset [\psi_r]\ et\ \psi \in \bar{\rho}\ et\ [\psi[\delta_r]] \subset [\psi']\}$
- $\forall \psi \in \bar{\rho}\ .\ Inv_{A_R/\sim}(\psi) = \bigvee_{\varphi \in Time(\psi)} \varphi$

Théorème 5.2. *Le STE d'un ATS-EAR et le STE du ATS-EAR quotient sont bisimilaires.*

Démonstration. Soit R un ensemble d'actions-règles. Nous allons montrer qu'il existe une relation de bisimulation entre leurs STEs respectifs Σ_{A_R} et $\Sigma_{A_R/\sim}$. En fait, nous allons montrer qu'il existe une relation d'équivalence, que nous notons $\sim_t$, et par rapport à laquelle $\Sigma_{A_R/\sim}$ est le STE quotient réduisant Σ_{A_R} . Pour définir la relation $\sim_t$, soient $e = (\omega, \theta)$ et $e' = (\omega', \theta')$ deux états de Σ_{A_R} :

$$e \sim_t e' \quad ssi \quad \omega \sim \omega'\ et\ \theta = \theta'$$

La relation $\sim_t$ correspond à la "temporisation" de $\sim$. Pour montrer que $\sim_t$ est une bisimulation nous distinguons les transitions d'écoulement du temps des transitions instantanées de Σ_{A_R}.
Soient $e = (\omega, \theta)$ et $e' = (\omega', \theta')$

- Pour les transitions d'écoulement du temps :
  si $e \sim_t e'$ et $e \xrightarrow{d} e + d$ alors $e' \xrightarrow{d} e' + d$ et $(e + d) \sim_t (e' + d)$ car $Inv(\omega) = Inv(\omega')$ d'après le corollaire 5.1.
- Pour les transitions instantanées :
  si $e \sim_t e'$ et $e \xrightarrow{l_r} (e[\delta_r], e[\lambda_r])$ telle que r est une action-règle , alors il existe une transition $e' \xrightarrow{l_r} (e'[\delta_r], e'[\lambda_r])$ et $(e[\delta_r], e[\lambda_r]) \sim_t (e'[\delta_r], e'[\lambda_r])$ car $\sim$ est une bisimulation et $e[\lambda_r] = e'[\lambda_r]$.

La relation de $\sim_t$ est à la fois une bisimulation entre Σ_{A_R} et lui même, elle est dite une auto-bisimulation. En réduisant le STE Σ_{A_R} par rapport à $\sim_t$, nous définissons le STE $\Sigma_{A_R}/\sim_t$. Nous remarquons que $\Sigma_{A_R}/\sim_t = \Sigma_{A_R/\sim}$. D'où Σ_{A_R} et $\Sigma_{A_R/\sim}$ sont bisimilaires.

□

Le résultat du théorème 5.2 est plus fort que celui publié dans [SDL01a] où nous réduisons l'ATS-EAR seulement par rapport à une relation d'équivalence observationnelle.

5.4 Exemple d'application

Dans cette section, nous allons illustrer la construction de l'ATS-EAR quotient. Pour décrire un ensemble d'actions-règles , il faut d'abord donner la description du domaine d'application qui consiste à spécifier les éléments des ensembles V, H et Ev (figure 5.8).

Nous spécifions le comportement partiel d'un commutateur téléphonique par un ensemble d'actions-règles décrites à la table 5.2. Le MSC de la figure 5.4 permet de faire abstraction de certains détails et montre une exécution possible du système. Nous visons par l'exemple courant seulement l'illustration de la construction du ATS-EAR quotient à partir d'un ensemble d'actions-règles. Ainsi nous ne tarderons pas ici à expliquer comment ces actions-règles sont été obtenues à partir des scénarios, nous verrons cela au chapitre 6.

Puisque toutes les actions-règles de la table 5.2 sont consistantes selon la définition 5.2, nous pouvons définir leur ATS-EAR et nous passons à la construction de leur ATS-EAR quotient.

Chaque case de la colonne $(*)$ dans la table 5.2 représente deux éléments qui sont calculés selon les expressions (5.4) et (5.5) respectivement et qui correspondent à l'action-règle de la même ligne que cette case.

$$\begin{aligned}
\psi_1 = &\ (A_status = IDLE \wedge A_signal = NONE)\\
\psi_2 = &\ (A_status = BUSY \wedge A_signal = NONE)\\
\psi_3 = &\ (A_status = BUSY \wedge A_signal = TONE)\\
\psi_4 = &\ (A_status = BUSY \wedge A_signal = DIALING(B))\\
\psi_5 = &\ (A_status = BUSY \wedge A_signal = DIALING(B)\\
&\wedge B_status = IDLE \wedge B_signal = NONE)
\end{aligned}$$

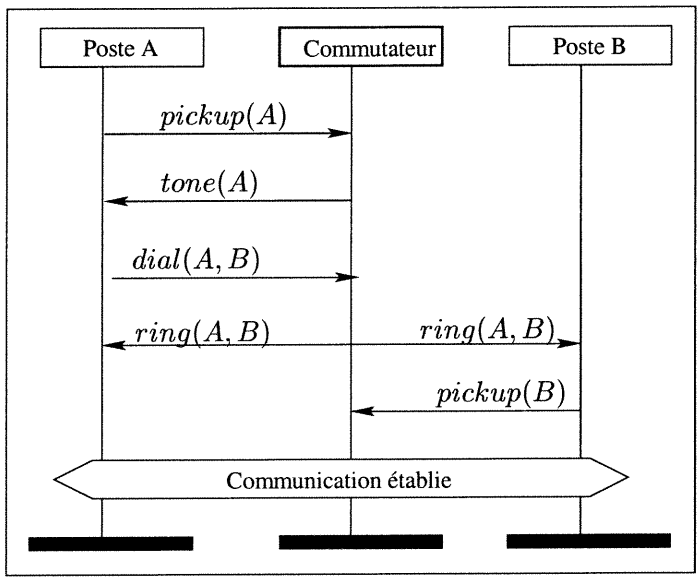


FIG. 5.7 – MSC représentant une exécution possible du commutateur téléphonique

$V = \{A\_status,\ A\_signal,\ B\_status,\ B\_signal\}$
$dom(A\_status) = \{BUSY,\ IDLE\}$
$dom(A\_signal) = \{NONE,\ TONE,\ BUSY\_TONE,\ DIALING(B),$
$ECHO\_RING(B),\ TALKING(B)\}$
$dom(B\_status) = \{BUSY\ , IDLE\};$
$dom(B\_signal) = \{NONE\ , TONE\ , BUSY\_TONE\ , RING(A)\ , TALKING(A)\}$
$H = \{h1,\ h2\}$
$Ev = \{pickup(A),\ pickup(B),\ send\_tone(A),\ busy\_tone(A),\ dialing(B)\}$

FIG. 5.8 – Description du domaine d'application pour le commutateur téléphonique

φ_r	l_r	ψ_r	φ'_r	δ_r	λ_r	φ''_r	$(*)$
$True$	$pickup(A)$	$A\_status{=}IDLE$, $A\_signal{=}NONE$	$True$	$A\_status{:=}BUSY$	$\{h2\}$	$h2{=}0$	ψ_1 ψ_2
$h2{=}0$	$send\_tone(A)$	$A\_status{=}BUSY$, $A\_signal{=}NONE$	$h2{=}0$	$A\_signal{:=}TONE$	$\{h1\}$	$h1{\leq}30$	ψ_2 ψ_3
$h1{\leq}30$	$dialing(B)$	$A\_status{=}BUSY$, $A\_signal{=}TONE$	$h1{<}30$	$A\_signal{:=}$ $DIALING(B)$	$\{h2\}$	$h2{=}0$	ψ_3 ψ_4
$h1{\leq}30$	$busy\_tone(A)$	$A\_status{=}BUSY$, $A\_signal{=}TONE$	$h1{=}30$	$A\_signal{:=}$ $BUSY\_TONE$	$\{h1\}$	$True$	ψ_3 ψ_8
$h2{=}0$	$ring(A,B)$	$A\_signal{=}$ $DIALING(B)$ $A\_status{=}BUSY$, $B\_status{=}IDLE$, $B\_signal{=}NONE$	$h2{=}0$	$A\_signal{:=}$ $ECHO\_RING(B)$, $B\_signal{:=}RING(A)$, $B\_status{:=}BUSY$,	$\{h2\}$	$h2{\leq}60$	ψ_5 ψ_6
$h2{\leq}60$	$pickup(B)$	$A\_signal{=}$ $ECHO\_RING(B)$, $B\_signal{=}RING(A)$, $A\_status{=}BUSY$, $B\_status{=}BUSY$,	$h2{<}60$	$A\_signal{:=}$ $TALKING$, $B\_signal{:=}$ $TALKING$	$\{h2\}$	$True$	ψ_6 ψ_7
$h2{=}60$	$busy\_tone(A)$	$A\_signal{=}$ $ECHO\_RING(B)$, $B\_signal{=}RING(A)$, $A\_status{=}BUSY$, $B\_status{=}BUSY$,	$h2{\leq}60$	$A\_signal{:=}$ $BUSY\_TONE$, $B\_signal{:=}NONE$, $B\_status{:=}IDLE$,	$\{h2\}$	$True$	ψ_6 ψ_9

TAB. 5.2 – Un ensemble d'actions-règles spécifiant le comportement du commutateur téléphonique

$$\begin{aligned}
\psi_6 = \ & (A_status = BUSY \wedge A_signal = ECHO_RING(B) \\
& \wedge B_status = BUSY \wedge B_signal = RING(A)) \\
\psi_7 = \ & (A_status = BUSY \wedge A_signal = TALKING(B) \\
& \wedge B_status = BUSY \wedge B_signal = TALKING(A)) \\
\psi_8 = \ & (A_status = BUSY \wedge A_signal = BUSY_TONE) \\
\psi_9 = \ & (A_status = BUSY \wedge A_signal = BUSY_TONE \\
& \wedge B_status = IDLE \wedge B_signal = NONE)
\end{aligned}$$

Selon les équations (5.4) et (5.5) nous avons :

$$\Psi_R = \{\psi_1, \psi_2, \psi_3, \psi_5, \psi_6\}$$

$$\Psi'_R = \{\psi_2, \psi_3, \psi_4, \psi_6, \psi_7, \psi_8, \psi_9\}$$

Nous calculons P selon l'équation (5.6) :

$$P = \{\psi_2, \psi_3, \psi_4, \psi_6, \psi_7, \psi_8, \psi_9, \psi_{10}\}$$

tel que,

$$\psi_{10} = \underbrace{\psi_1 \vee \psi_2 \vee \psi_3 \vee \psi_4 \vee \psi_5 \vee \psi_6 \vee \psi_7 \vee \psi_8 \vee \psi_9}_{Q=[\Psi_R]\cup[\Psi'_R]} \wedge \neg \underbrace{(\psi_2 \vee \psi_3 \vee \psi_4 \vee \psi_6 \vee \psi_7 \vee \psi_8)}_{[\Psi'_R]}$$

En simplifiant ψ_{10} nous obtenons, $\psi_{10} = \psi_1$ d'où

$$P = \{\psi_2, \psi_3, \psi_4, \psi_6, \psi_7, \psi_8, \psi_9, \psi_1\}$$

Nous calculons maintenant la partition initiale $\rho_o = part(P)$ comme c'est défini par les équations (5.9) et (5.8). Pour obtenir ρ_o il suffit de partitionner $[\psi_4]$ et $[\psi_8]$. Nous partitionnons $[\psi_4]$ par rapport à $[\psi_5]$ en $[\psi_{41}]$ et $[\psi_{42}]$,

$$\psi_{41} = \psi_4 \wedge \psi_5 = \psi_5 \text{ et } \psi_{42} = \psi_4 \wedge \neg\psi_5$$

De même, nous partitionnons $[\psi_8]$ par rapport à $[\psi_9]$ en $[\psi_{81}]$ et $[\psi_{82}]$,

$$\psi_{81} = \psi_8 \wedge \psi_9 = \psi_9 \text{ et } \psi_{82} = \psi_8 \wedge \neg\psi_9$$

D'où,

$$\rho_o = \{\psi_1, \psi_2, \psi_3, \psi_{41}, \psi_{42}, \psi_6, \psi_7, \psi_{81}, \psi_{82}\}$$

Nous appliquons maintenant la procédure de *ω-minimisation* (figure 5.4). Nous représentons les valeurs des données de la procédure de *ω-minimisation* (figure 5.3) après chaque partitionnement d'une classe par la fonction *split*.

Initialisation : $\rho = \rho_o$, $Stb = \emptyset$,

$split(\rho, \psi_3)$ **:** Nous choisissons ψ_3. $[\psi_3]$ est partitionnée en $[\psi_{31}]$ et $[\psi_{32}]$:

$$\begin{aligned}\psi_{31} =&(\{A\_status = BUSY \wedge A\_signal = TONE \\ &\wedge B\_status = IDLE \wedge B\_signal = NONE) \\ \psi_{32} =&(\{A\_status = BUSY \wedge A\_signal = TONE \\ &\wedge (B\_status = BUSY \vee B\_signal \neq NONE))\end{aligned}$$

$$Stb = \emptyset, \rho = \{\psi_1, \psi_2, \psi_{31}, \psi_{32}, \psi_{41}, \psi_{42}, \psi_6, \psi_7, \psi_{41}, \psi_{42}\}$$

$split(\rho, \psi_2)$ **:** ψ_2 est partitionnée en ψ_{21} et ψ_{22} :

$$\begin{aligned}\psi_{21} =&(\{A\_status = BUSY \wedge A\_signal = NONE \\ &\wedge B\_status = IDLE \wedge B\_signal = NONE\}) \\ \psi_{22} =&(\{A\_status = BUSY \wedge A\_signal = NONE \\ &\wedge (B\_status = BUSY \vee B\_signal \neq NONE))\end{aligned}$$

$$Stb = \emptyset, \rho = \{\psi_1, \psi_{21}, \psi_{22}, \psi_{31}, \psi_{32}, \psi_{41}, \psi_{42}, \psi_6, \psi_7, \psi_{41}, \psi_{42}\}$$

$split(\rho, \psi_3)$ **:** ψ_1 est partitionnée en ψ_{11} et ψ_{12} :

$$\begin{aligned}\psi_{11} =&(\{A\_status = IDLE \wedge A\_signal = NONE \\ &\wedge B\_status = IDLE \wedge B\_signal = NONE) \\ \psi_{12} =&(\{A\_status = IDLE \wedge A\_signal = NONE \\ &\wedge (B\_status = BUSY\} \vee B\_signal \neq NONE))\end{aligned}$$

$$Stb = \emptyset, \rho = \{\psi_{11}, \psi_{12}, \psi_{21}, \psi_{22}, \psi_{31}, \psi_{32}, \psi_{41}, \psi_{42}, \psi_6, \psi_7, \psi_{41}, \psi_{42}\}$$

A ce stade toutes les classes sont stables.

$$\bar{\rho} = \{\psi_{11}, \psi_{12}, \psi_{21}, \psi_{22}, \psi_{31}, \psi_{32}, \psi_{41}, \psi_{42}, \psi_6, \psi_7, \psi_{41}, \psi_{42}\}$$

À partir de la table 5.2, nous pouvons déterminer $Time(\psi)$ pour toute classe ψ appartenant à $\bar{\rho}$. Dans le graphe 5.9, les contraintes temporelles ayant un fond gris représente les invariants respectifs des places de l'automate.

L'automate de la figure 5.9 représente le graphe de l'ATS-EAR quotient associé à l'ensemble d'actions-règles de la table 5.2. Nous avons regroupé certaines places de l'ATS-EAR quotient en un noeud du graphe de la figure 5.9 pour montrer le partitionnement qu'ont subit les éléments de l'ensemble P (équation (5.6)) par les fonctions $split$ et $part$.

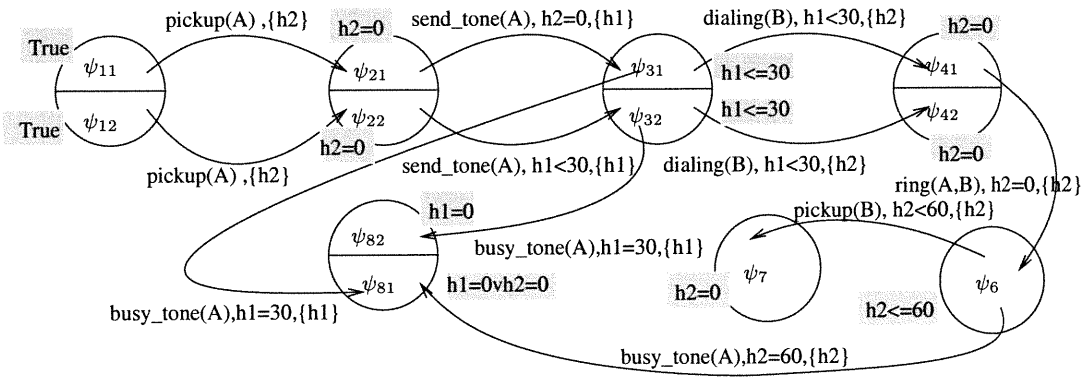


FIG. 5.9 – ATS-EAR quotient des actions-règles de la table 5.2

5.5 Conclusion

Nous avons présenté un modèle pour exprimer la spécification d'un système réactif temps-réel. Elle se compose d'une description statique et d'une description dynamique. La description statique représente une concrétisation des propriétés du système sous forme d'éléments du langage qui sont utilisés pour décrire son comportement. La description statique représente la description du domaine d'application, elle est constituée de trois éléments : les horloges du système, ses variables discrètes et ses événements.

Bien que le comportement du système soit supposé être décrit par un ensemble d'*actions-règles*, celui -ci provient d'un ensemble de scénarios comme nous allons le voir dans les chapitres suivants. La sémantique d'un ensemble d'actions-règles a été exprimée par un système de transitions étiquetées pour lequel nous avons construit un automate temporisé sécuritaire. Afin de contrer l'explosion combinatoire dans ces automates, nous les avons réduits en utilisant une relation de bisimulation.

En résumé, dans ce chapitre nous avons montré comment construire un automate temporisé sécuritaire à partir d'actions-règles. Le chapitre suivant définit notre modèle de scénario dont la forme canonique est un ensemble d'actions-règles.

Chapitre 6

Modélisation des scénarios du comportement d'un système réactif temps-réel

Comme nous l'avons déjà mentionné au chapitre 1, nous visons à générer de manière automatique la spécification d'un système réactif temps-réel à partir de spécifications partielles qui sont saisies sous forme de scénarios. L'objectif de ce chapitre est la définition d'un nouveau formalisme pour la modélisation des scénarios auquel nous associons une sémantique formelle qui sera préservée pendant toutes les opérations d'intégration qui mènent à la génération automatique de la spécification du système. Remarquons qu'une sémantique formelle est nécessaire pour que le processus d'intégration des scénarios soit automatique.

A partir d'un scénario nous allons extraire un ensemble d'actions-règles représentant sa forme canonique. Ainsi, la sémantique formelle d'un scénario est basée sur la sémantique de ses actions-règles. Une fois que nous aurons défini les actions-règles d'un scénario, le processus peut être enchaîné en lui associant un ATS-EAR qui est basé sur l'ensemble de ses actions-règles . Cet ATS-EAR est une spécification partielle qui modélise le comportement du scénario.

Dans le plan du chapitre, la section 6.1 définit le modèle et la syntaxe des scénarios. La section 6.2 présente la forme canonique d'un scénario sous forme d'actions-règles auxquelles, en fin de compte, nous associons un ATS. La section 6.3 traite un exemple d'application.

6.1 Modélisation d'un scénario

Un scénario est une description partielle du comportement du système. L'aspect partiel de la description est dû à deux raisons. La première est reliée l'exécution du scénario qui ne peut être effectuée que dans un contexte d'utilisation qui satisfait certaines pré-conditions. La seconde raison est le fait que le système peut aussi permettre d'autres comportements qui ne sont pas décrits dans le scénario courant mais qui peuvent faire l'objet d'autres scénarios.

La section courante décrit l'information qu'un scénario véhicule. Ensuite, elle présente le modèle formel d'un scénario.

6.1.1 Information véhiculée par un scénario

Même si le comportement décrit par un scénario est un comportement partiel, le scénario doit comporter assez d'information pour qu'il soit intégré dans un contexte global. En général, un scénario de comportement d'un système réactif est décrit sous forme d'une séquence continue de *stimulation/réaction*. Cette forme, couplant les entrées et les sorties, est bien adaptée à la spécification des composantes matérielles. Nous avons adopté un modèle similaire dans [SDLV00].

Cependant les systèmes réactifs peuvent avoir des spécifications plus complexes sous forme de séquences d'actions où chaque action est soit une stimulation (une entrée), soit une réaction (une sortie). Ce modèle est plus général car il évite de coupler une stimulation à une réaction et permet d'avoir des interactions plus libres. Ainsi, il est plus raisonnable de considérer qu'un scénario est composé de séquences d'actions [SDL01a]. Par conséquent, de manière informelle, le modèle d'un scénario comporte nécessairement :

- un contexte d'exécution exprimé sous forme de contraintes sur l'état du système représentant les pré-conditions du scénario,
- et une séquence d'événements. Le contexte d'exécution est une pré-condition du premier événement de la séquence. Ensuite, l'état du système résultant après un événement constitue les pré-conditions de l'événement suivant dans la séquence du scénario.

L'action est l'unité élémentaire composant un scénario dans la mesure où chaque événement est décrit sous forme d'une action dont la description doit porter toute l'information nécessaire pour l'insérer dans la spécification globale du système. Cette information concerne les aspects temporels et non temporels.

6.1.2 Définition du modèle d'un scénario

Nous rappelons que dans le chapitre précédent, nous avons associé au système trois ensembles H, V et Ev où H représente un ensemble d'horloges, V désigne un ensemble de variables discrètes et Ev est un ensemble d'événements.

Chacune des actions d'un scénario, décrit le changement que subit l'état du système soit après l'exécution de cette action soit à cause de l'écoulement d'un délai. Nous commençons par définir la syntaxe des actions des scénarios ensuite nous définissons le modèle des scénarios comme étant une structure arborescente d'actions.

Définition 6.1. *$a = (\psi_a, \varphi_a, \varphi'_a, lab_a, \delta_a, \lambda_a)$ est une actions d'un scénario ssi :*

- *$\psi_a \in \Psi(V)$ est une contrainte de variables du système qui doit être satisfaite avant l'exécution de l'action.*
- *$\varphi_a \in \Phi(H)$ est une contrainte d'horloges qui joue le rôle d'invariant sur l'état avant l'exécution de l'action.*
- *$l_a \in Ev$ est une étiquette qui permet l'observation de l'événement marquant l'exécution de l'action.*
- *$\varphi'_a \in \Phi(H)$ est un garde temporel qui permet l'exécution de l'action.*
- *$\delta_a \in \Delta(V)$ est une affectation de variables décrivant la mise à jour des valeurs des variables après l'exécution de l'action.*
- *λ_a est une assignation d'horloges qui a lieu après l'exécution de l'action.*

Nous notons aussi $sc.a = (\psi_{sc.a}, \varphi_{sc.a}, \varphi'_{asc.}, lab_{sc.a}, \delta_{sc.a}, \lambda_{sc.a})$ l'action a du scénario sc s'il y a un risque de confusion lors de la manipulation de plusieurs scénarios. Nous modélisons la spécification d'un scénario par un ensemble de séquences d'actions qui est structuré sous forme d'un arbre que nous schématisons à la figure 6.1.

Définition 6.2. *Un scénario sc est un arbre $(\mathcal{N}, \longmapsto)$ où :*

- *$\mathcal{N} = \mathcal{N}_p \cup \mathcal{N}_s$ tel que $\mathcal{N}_p$ et $\mathcal{N}_s$ sont respectivement les ensembles des noeuds principaux et secondaires du scénario sc. Nous posons $\mathcal{N}_p = [N_0, N_1, \ldots, N_{n-1}, N_n]$ avec $n+1 = |\mathcal{N}_p|$.*
- *$\longmapsto \subset \mathcal{N} \times \mathcal{N}$ est la relation père fils entre les noeuds de l'arbre du scénario. Pour chaque couple de noeuds $(N, N') \in \longmapsto$, le noeud N' est le noeud fils de N. De plus, (N, N') est associé à une action "a" du scénario que nous notons $N \overset{a}{\longmapsto} N'$.*

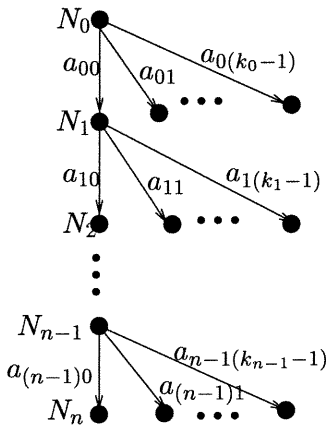


FIG. 6.1 – Arbre d'un scénario

Pour tout noeud principal $N_i \in \mathcal{N}_p$ *tel que* $0 \leq i \leq n-1$, N_i *est un ensemble ordonné d'actions de la forme* $N_i = [a_{i0}, \ldots, a_{i(k_i-1)}]$ *avec* $k_i = |N_i|$ *et* $N_i \xmapsto{a_{i0}} N_{i+1}$. *Pour* $i = n$, *nous avons* $|N_i| = 0$.

De manière similaire aux actions, un noeud $N \in \mathcal{N}$ du scénario sc peut être noté $sc.N$ si une confusion sur le scénario est possible.

L'action a_{i0} est dite l'action principale du noeud principal N_i car elle relie deux noeuds principaux successifs N_i et N_{i+1}. En outre les feuilles de l'arbre d'un scénarios sont constituées des noeuds secondaires et du dernier noeud principal. Les chemins qui mènent de la racine de l'arbre d'un scénario aux feuilles, représentent les différentes exécutions possibles du scénario. La conception d'un scénario est souvent motivée par un but que le scénario doit réaliser en s'exécutant. L'exécution du scénario qui passe par toutes les actions principales représente "une exécution complète" car elle est sensée accomplir le but pour lequel ce scénario a été conçu. Dans un noeud principal N_i du scénario, les actions a_{ij}, avec $1 \leq j < k_i$, représentent des alternatives qui peuvent être exécutée à la place de l'action principale a_{i0} ; si celle-ci ne peut pas s'exécuter. Les actions alternatives spécifient comment le système agirait lorsque l'environnement ne permet pas l'exécution de l'action principale du noeud. Ces alternatives permettent ainsi d'éviter le blocage du système. Par exemple, si le scénario spécifie à travers une action principale la réception d'un acquittement de message avant un certain délai, l'une des alternatives de cette action princi-

pale peut être le traitement de l'expiration de ce délai.

L'intégration des scénarios en un modèle global doit préserver la sémantique des scénarios. Dans la section suivante, nous définissons cette sémantique.

6.2 Sémantique associée à un scénario

Après avoir donné la représentation formelle du modèle d'un scénario, nous allons définir sa sémantique. Afin de faciliter l'expression de cette sémantique, nous transformons le scénario sous une forme canonique que nous représentons sous forme d'un ensemble d'actions-règles. Les actions-règles d'un scénario définissent un système de transitions étiquetées (STE-EAR) qui représente la sémantique du scénario.

6.2.1 L'actions-règles d'une action d'un scénario

La structure d'un scénario (figure 6.1) est modélisée sous forme d'une séquence linéaire de noeuds principaux. Chaque noeud principal représente le *contexte* où ses actions peuvent être exécutées. Un *contexte* est un regroupement des états du système qui satisfont un couple de contraintes composé d'une contrainte d'horloges et d'une contrainte de variables. La contrainte d'horloges du *contexte* représente la condition d'activité du noeud. Le contexte d'un noeud principal dépend des contraintes de son action principale et du contexte qui résulte de l'exécution de l'action principale du noeud principal précédent. L'exécution d'une action principale a_0 du noeud principal N_i rend possible l'exécution d'une action appartenant au noeud principal cible N_{i+1}. C'est à dire que l'exécution de a_{i0} crée le *contexte* de N_{i+1} qui est nécessaire pour l'exécution de ses actions. L'exécution du scénario s'arrête lorsque l'exécution d'une action mène à un noeud cible feuille. Pour transformer les actions d'un scénario en actions-règles indépendantes, nous allons associer à chaque noeud principal N_i deux contraintes $\psi_{N_i} \in \Psi(V)$ et $\varphi_{N_i} \in \Phi^+(H)$ qui définissent son contexte. Pour formuler les contraintes ψ_{N_i} et φ_{N_i}, nous distinguons trois cas selon l'index i :

– Pour $i = 0$,

$$N_0 : \begin{cases} \psi_{N_0} = \psi_{a_{00}} \\ \varphi_{N_0} = \bigwedge\limits_{0 \leq k \leq k_0 - 1} \varphi_{a_{0k}} \end{cases} \tag{6.1}$$

– Pour $1 \leq i \leq n-1$,

$$N_i : \begin{cases} \psi_{N_i} = (\psi_{N_{i-1}}[\delta_{a_{(i-1)0}}] \wedge \psi_{a_{i0}}) \\ \varphi_{N_i} = \bigwedge\limits_{1 \leq k \leq k_i - 1} \varphi_{a_{ik}} \end{cases} \tag{6.2}$$

– Pour $i = n$,

$$N_n : \begin{cases} \psi_{N_n} = \psi_{N_{n-1}}[\delta_{a_{(n-1)0}}] \\ \varphi_{N_n} = (\varphi_{N_{n-1}} \wedge \varphi'_{a_{(n-1)0}})[\lambda_{a_{(n-1)0}}] \end{cases} \tag{6.3}$$

Les formules (6.1), (6.3) et (6.2) expriment respectivement les contraintes du contexte du premier noeud principal, celles du dernier noeud principal et celles des autres noeuds principaux.

L'équation (6.1) indique que la contrainte de variables qui associée au contexte du premier noeud principal d'un scénario, est exactement la même contrainte (de variable) que celle qui est spécifiée par l'action principale du noeud en question. Ainsi l'état du système doit au moins satisfaire la contrainte de variables de l'action principale du noeud pour que les autres actions du noeud, c'est à dire les alternatives, soient exécutables. Cette remarque reste vraie même pour les autres noeuds principaux du scénario.

Par contre, la contrainte d'horloges du contexte du premier noeud principal est constituée par la conjonction des invariants de toutes les actions du noeud. La contrainte d'horloges du contexte représente ainsi une condition d'activité qui est commune à toutes les actions du noeud.

Supposons que N_i est un noeud principal qui n'est ni le premier noeud du scénario ni le dernier. L'expression de ψ_{N_i} dans l'équation (6.2) montre une propagation de la contrainte de variables associée au contexte de N_{i-1} à travers son action principale $a_{(i-1)0}$. Ainsi ψ_{N_i} est donnée par la conjonction de la contrainte de variables de l'action principale de N_i et de la contrainte de variables qui résulte de l'exécution de l'action $a_{(i-1)0}$ dans son contexte décrit par $\psi_{N_{i-1}}$. Nous assurons ainsi que l'exécution d'une action qui appartient au noeud N_i, n'est possible que si l'action $a_{(i-1)0}$ du noeud N_{i-1} a été déjà exécutée. La contrainte d'horloges du contexte du noeud N_i est calculée de manière similaire à celle du premier noeud principal.

Les contraintes du contexte du dernier noeud principal d'un scénario (équations (6.3)) sont exactement les contraintes que satisfait l'état du système après l'exécution de l'action principale

du noeud précédent dans son contexte.

Pour uniformiser la notion de contraintes associées aux noeuds principaux d'un scénario, nous définissons aussi les contraintes[1] du contexte d'un noeud secondaire N tel que $N_i \xmapsto{a_{ij}} N$ par :

$$N : \begin{cases} \psi_N = \psi_{N_i}[\delta_{a_{ij}}] \\ \varphi_N = (\varphi_{N_i} \wedge \varphi'_{a_{ij}})[\lambda_{a_{ij}}] \end{cases} \tag{6.4}$$

Les expressions des contraintes dans les équations (6.4) sont communes à tous les noeuds feuilles d'un scénario.

L'exécution d'une action dépend des contraintes du contexte de son noeud principal. En regroupant la description d'une action et les contraintes du contexte de son noeud principal nous constituons une action-règle qui comporte toute l'information sur l'exécution de cette action. Nous définissons l'action-règle r qui capture la sémantique de l'action a_{ij} du noeud principal N_i :

Définition 6.3. *$r = (\psi_r, \varphi_r, l_r, \varphi'_r, \delta_r, \lambda_r, \varphi''_r)$ est l'action-règle de l'action a_{ij} du noeud principal N_i ssi :*

- *$\psi_r = \psi_{N_i} \wedge \psi_{a_{ij}}$ est la contrainte sur les variables à satisfaire avant l'exécution de l'action-règle.*
- *$\varphi_r = \varphi_{N_i}$ est la condition d'activité de l'action-règle.*
- *$l_r = l_{a_{ij}}$*
- *$\varphi'_r = \varphi'_{a_{ij}}$*
- *$\delta_r = \delta_{a_{ij}}$*
- *$\lambda_r = \lambda_{a_{ij}}$*
- *Pour φ''_r, deux cas sont à distinguer selon que le noeud cible après l'exécution de a_{ij} est un noeud feuille de l'arbre du scénario ou non :*
  - *(a) si $0 \leq i < n-1$ et $(j = 0)$ alors $\varphi''_r = \varphi_{N_{i+1}}$*
  - *(b) si $((0 \leq i \leq n-1$ et $0 < j \leq k_i - 1)$ ou $(i = n-1$ et $j = 0))$ alors $\varphi''_r = (\varphi_{N_i} \wedge \varphi'_{a_{ij}})[\lambda_{aij}]$*

Dans la définition 6.3, l'expression de ψ_r est égale à la conjonction de ψ_{N_i} et $\psi_{a_{ij}}$ car pour exécuter l'action a_{ij} il faut satisfaire la contrainte du contexte du noeud de a_{ij} et la contrainte de variables qui est spécifiée par a_{ij}. Par ailleurs, l'invariant d'une action-règle qui correspond

[1] Ces contraintes seront utilisées pour l'intégration des scénarios dans les chapitres suivants

à une action est exactement égale à l'invariant du contexte du noeud principal de cette action. L'action d'un scénario ne comporte pas d'information sur l'invariant de l'état du système après son exécution, cependant cette information peut être déduite à partir du séquencement du scénario en considérant l'invariant de l'action suivante si elle existe. L'invariant de l'état du système après l'exécution d'une action est aussi représenté dans son action-règle, ainsi dans l'expression de φ''_r (définition 6.3) nous distinguons deux cas selon que cette action possède une action suivante dans le scénario ou pas. Dans le premier cas, c'est l'invariant du contexte du noeud principal suivant. Dans le second cas, nous instancions l'invariant φ'_r à tout l'espace temporel accessible après l'exécution de r.

Exemple 6.1. *Dans cet exemple nous illustrons le calcul des actions-règles du scénario donné par la table 6.1.*

a	φ_a	l_a	ψ_a	φ'_a	δ_a	λ_a
a_{00}	$h \leq 20$	$event_1$	$v_1 \leq 6$	$h<20$	$v_1 := v_1 + 1$	$\{h\}$
a_{01}	$h \leq 20$	$event_2$	$v_2 \leq 3$	$h=20$	$v_1 := 1$	$\{h\}$
a_{10}	$True$	$event_3$	$v_2 = 0$	$h<30$	$v_1 := v_1 - 1, v_2 := v_2 + 1$	$\{h\}$
a_{11}	$True$	$event_4$	$True$	$h \geq 30$	$v_1 := 0, v_2 := 0$	$\{h\}$

TAB. 6.1 – Tableau décrivant un scénario

En adoptant les notations des équations (6.1) (6.2) *et* (6.3)*, nous avons :*

- $\varphi_{N_0} = (h \leq 20)$ *et* $\psi_{N_0} = (v_1 \leq 6)$
- $\varphi_{N_1} = (True)$ *et* $\psi_{N_1} = (v_1 \leq 7 \wedge v_2 = 0)$
- $\varphi_{N_2} = (h = 0)$ *et* $\psi_{N_2} = (v_1 \leq 6 \wedge v_2 = 1)$

D'après la définition 6.3, nous donons à la table 6.2 l'ensemble des actions-règles du scénario décrit par la table 6.1

6.2.2 La consistance d'un scénario

La définition de la consistance d'un scénario se base sur celle de la consistance de ses actions-règles selon la définition 5.2.

Définition 6.4. *Un scénario est dit consistant ssi toutes ses actions-règles sont consistantes.*

φ_r	l_r	ψ_r	φ'_r	δ_r	λ_r	φ''_r
$h \leq 20$	$event_1$	$v_1 \leq 6$	$h < 20$	$v_1 := v_1 + 1$	$\{h\}$	$True$
$h \leq 20$	$event_2$	$v_1 \leq 6 \wedge v_2 \leq 3$	$h=20$	$v_1 := 1$	$\{h\}$	$h = 0$
$True$	$event_3$	$v_1 \leq 7 \wedge v_2 = 0$	$h<30$	$v_1 := v_1 - 1, v_2 := v_2 + 1$	$\{h\}$	$h = 0$
$True$	$event_4$	$v_1 \leq 7 \wedge v_2 = 0$	$h \geq 30$	$v_1 := 0, v_2 := 0$	$\{h\}$	$h = 0$

TAB. 6.2 – Ensemble des actions-règles du scénario de la table 6.1

La consistance d'un scénario n'implique pas forcément que la description de spécification que fournit le scénario est correcte. Un scénario consistant veut dire seulement que les actions-règles résultantes des actions du scénario sont consistantes et donc exécutables. La vérification d'une spécification globale du système permet de détecter certaines erreurs qui ne rentrent pas dans le cadre de la consistance d'un scénario telle que définie par la définition 6.4. Un exemple de telles erreurs peuvent être, le non-déterminisme ou l'interaction de services qui résultent d'une contradiction entre un scénario et lui même ou bien entre deux ou plusieurs scénarios. La contradiction d'un scénario avec lui-même représente le cas où un scénario comporte deux actions qui se contredisent.

6.2.3 La forme canonique d'un scénario

Les actions-règles d'un scénario représentent sa forme canonique. Elles sont indépendantes les unes des autres et chacune comporte toute l'information concernant les changements de l'état du système lors de son exécution ou bien pendant l'écoulement du temps. Bien que chaque action d'un scénario donne lieu à une action-règle selon la définition 6.3, nous ne pouvons retenir toutes les actions-règles du scénario pour formuler sa forme canonique. En éliminant les actions-règles redondantes, nous constituons un ensemble d'actions-règles qui représente la forme canonique d'un scénario. La redondance des actions-règles se manifeste lorsque le comportement que décrit une action-règle, est déjà été décrit de manière plus générale, dans le scénario, par une autre action-règle où la première est "incluse". Nous définissons l'inclusion entre deux actions-règles par le fait que le comportement décrit par l'une soit inclus dans le comportement décrit par l'autre action-règle.

Définition 6.5. *Soient r_1 et r_2 deux actions-règles. Nous notons que r_1 est incluse dans r_2 par $r_1 \subset r_2$ ssi*

(1) $l_{r_1} = l_{r_2}$

(2) $\psi_{r1} \wedge \psi_{r2} = \psi_{r1}$

(3) $\varphi_{r1} \wedge \varphi_{r2} = \varphi_{r1}$

(4) $\varphi'_{r1} \wedge \varphi'_{r2} = \varphi'_{r1}$

(5) $\delta_{r_1} = \delta_{r_2}$

(6) $\lambda_{r_1} = \lambda_{r_2}$

Nous désignons par $R(sc)$ la forme canonique du scénario sc. Elle est représentée par l'ensemble de toutes les actions-règles non redondantes du scénario sc. La construction de $R(sc)$ consiste à comparer deux à deux les actions-règles issues des actions du scénarios sc et en cas d'inclusion d'une action-règle dans une autre, seule l'action-règle la plus générale est à garder dans $R(sc)$. L'algorithme pour calculer la forme canonique d'un scénario est décrit à la figure 6.2.

fonction $forme\_canonique(sc)$

Input : sc un scénario

Output : $R(sc)$ forme canonique du scénario sc

$R(sc) := \emptyset$

pour chaque r action-règle de sc **faire**

si $(\exists r' \in R(sc)\ .\ r' \subset r)$

alors $R(sc) := (R(sc) \backslash \{r'\}) \cup \{r\}$

sinon $R(sc) := R(sc) \cup \{r\}$

retourner $R(sc)$

FIG. 6.2 – Fonction de calcul de la forme canonique d'un scénario

6.2.4 Automate temporisé sécuritaire d'un scénario

Dans le chapitre précédent, nous avons associé à un ensemble d'actions-règles un STE-EAR qui définit sa sémantique opérationnelle de manière globale. Dans le cas particulier où l'ensemble d'actions-règles considéré, représente la forme canonique d'un scénario, son STE-EAR permet de

définir la sémantique opérationnelle globale du scénario en question. Le LTS d'un scénario constitue une partie d'un LTS encore inconnu qui modélise le système.

Définition 6.6. *Soit sc un scénario de comportement du système. Le système de transitions étiquetées du scénario sc est le STE-EAR basé sur sa forme canonique $R(sc)$.*

Nous rappelons que le STE-EAR d'un ensemble d'actions-règles R quelconque est noté Σ_R ainsi nous notons $\Sigma_{R(sc)}$ le STE-EAR du scénario *sc*.

D'après le théorème 5.1, nous pouvons associer un ATS-EAR à un ensemble d'actions-règles consistantes, ainsi nous associons un ATS à un scénario consistant.

Définition 6.7. *Nous appelons automate temporisé sécuritaire du scénario consistant sc, l'ATS-EAR basé sa forme canonique $R(sc)$ et noté $A_{R(sc)}$.*

Le ATS-EAR $A_{R(sc)}$ représente une forme finie et compacte modélisant le scénario *sc*. L'automate temporisé d'un scénario peut être utilisé afin de vérifier si le scénario est correct. La vérification de l'ATS-EAR du scénario (par abus de langage) permet seulement de conclure, en cas de succès, que le scénario en tant qu'entité isolée ne comporte pas d'erreur. Nous appelons cette vérification *vérification intra-scénario*. Nous n'excluons pas qu'un scénario qui passe l'intra-vérification, puisse avoir des erreurs ou des conflits par rapport à d'autres scénarios. Les erreurs d'un scénario par rapport à d'autres peuvent être détectées par la vérification *inter-scénarios* qui est en fait la vérification de la spécification globale du système.

6.3 Exemple d'application

Dans cette section, nous illustrons la modélisation d'un scénario à partir d'une description en langue naturelle. Le scénario traité décrit le comportement d'un commutateur téléphonique simplifié. Nous supposons que ce commutateur compte deux postes téléphoniques A et B. Nous traitons un scénario de l'établissement de la communication entre le poste A et le poste B qui est décrit par le texte suivant :

Scénario d'établissement d'une communication

"Lorsque le poste A est libre, si le commutateur reçoit le message $pickup(A)$ alors il envoie la tonalité au poste A. Si le poste A appelle B avant 30 unité de temps

alors le commutateur envoie la sonnerie si le poste B est libre. Mais si le poste A ne fait rien pendant 30 unité de temps alors le commutateur lui envoie $busy\_tone(A)$. Lorsque le terminal du poste B est entrain de sonner, si celui-ci décroche avant 60 unité de temps, le commutateur établit la communication sinon il l'annule et envoie $busy\_tone(A)$."

Pour modéliser le scénario ci-dessus, il faut commencer par extraire la description du domaine d'application à partir du texte fourni. Le but est de déterminer l'ensemble des variables discrètes du système et leurs domaines, le nombre d'horloges nécessaire et un alphabet pour étiqueter les événements du système. La modélisation du système donne lieu aux descriptions respectives de V, H et Ev qui sont représentées à la figure 5.8 page 72.

Afin de comprendre et d'analyser le comportement du système, nous traduisons le scénario donné ci-dessus sous forme d'un MSC représenté à la figure 6.3. Nous proposons à la table 6.3 une modélisation du scénario de l'établissement d'une communication. L'arbre du scénario peut être déduit en considérant les indices des actions à la colonne "a" de la table 6.3. La forme canonique de ce scénario est exactement l'ensemble d'actions-règles décrit par la table 5.2 au chapitre précédent. Nous pouvons maintenant déduire l'origine de l'ensemble d'actions-règles de la table 5.2. Ce scénario est consistant car toutes les actions-règles de sa forme canonique sont consistantes. L'ATS-EAR quotient de ce scénario est celui représenté à la figure 5.9 page 76.

Remarque 6.1. *Dans cet exemple, nous avons utilisé deux horloges dans la modélisation du scénario d'établissement de la communication bien qu'une seule horloge aurait suffit. Dans des cas plus complexes, un algorithme pour la minimisation du nombre d'horloges [DY96] pourrait être utilisé.*

6.4 Conclusion

Dans ce chapitre nous avons défini un nouveau modèle de scénarios dont la sémantique est représentée par sa forme canonique sous forme d'actions-règles. Par conséquent l'automate temporisé sécuritaire d'un scénario est déduit de ses actions-règles. Une méthode intuitive d'intégration des scénarios serait de construire un automate temporisé sécuritaire à partir des actions-règles d'un

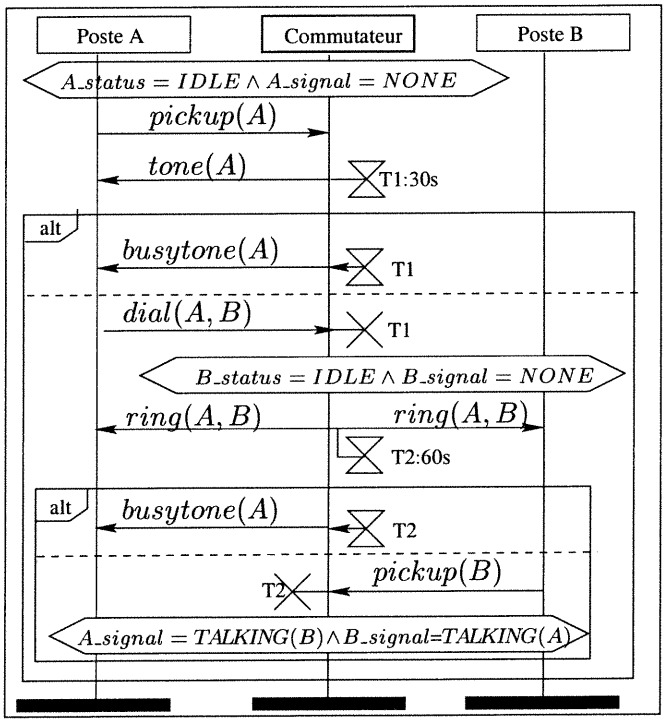


FIG. 6.3 – MSC représentant le scénario d'établissement d'une communication

a	φ_a	l_a	ψ_a	φ'_a	δ_a	λ_a
a_{00}	$True$	$pickup(A)$	$(A\_status{=}IDLE)$ $\wedge(A\_signal{=}NONE)$	$True$	$A\_status{:=}BUSY$	$\{h2\}$
a_{10}	$h2{=}0$	$send\_tone(A)$	$True$	$h2{=}0$	$A\_signal{:=}TONE$	$\{h1\}$
a_{20}	$h1{\leq}30$	$dialing(B)$	$True$	$h1{<}30$	$A\_signal{:=}$ $DIALING(B)$	$\{h2\}$
a_{21}	$h1{\leq}30$	$busy\_tone(A)$	$True$	$h1{=}30$	$A\_signal{:=}$ $BUSY\_TONE$	$\{h1\}$
a_{30}	$h2{=}0$	$ring(A,B)$	$(B\_status{=}IDLE)\wedge$ $(B\_signal{=}NONE)$	$h2{=}0$	$A\_signal{:=}$ $ECHO\_RING(B)$, $B\_signal{:=}RING(A)$, $B\_status{:=}BUSY$,	$\{h2\}$
a_{40}	$h2{\leq}60$	$pickup(B)$	$True$	$h2{<}60$	$A\_signal{:=}$ $TALKING$, $B\_signal{:=}$ $TALKING$	$\{h2\}$
a_{41}	$h2{=}60$	$busy\_tone(A)$	$True$	$h2{\leq}60$	$A\_signal{:=}$ $BUSY\_TONE$, $B\_signal{:=}NONE$, $B\_status{:=}IDLE$,	$\{h2\}$

TAB. 6.3 – Scénario d'établissement d'une communication

ensemble de scénarios. Cette méthode d'intégration fait partie des méthodes d'intégration implicites auxquelles nous consacrons le chapitre suivant.

Chapitre 7

Intégration implicite des scénarios temporisés

Au chapitre précédent nous avons défini notre modèle des scénarios et nous lui avons associé une sémantique formelle. Ainsi chaque scénario possède une interprétation unique. Par ailleurs, le comportement d'un système peut être décrit par plusieurs scénarios dont chacun représente une spécification partielle. Une spécification est une description exhaustive et formelle de tous les comportements possibles d'un système. Le regroupement des comportements des scénarios en une seule spécification représente la tâche d'intégration des scénarios. L'intégration de tous les scénarios possibles donne lieu à une spécification du système qui est exprimée dans un formalisme cible. La spécification résultante de l'intégration des scénarios devrait préserver leurs sémantiques. Vu la relation directe entre la sémantique de nos scénarios et celle des automates temporisés sécuritaires, il va de soi que ce modèle soit notre formalisme cible de l'intégration des scénarios.

Nous consacrons ce chapitre à l'intégration implicite des scénarios. Le terme *intégration implicite* exprime le fait que les scénarios sont regroupés sans aucune contrainte additionnelle sur l'ordre de leurs exécutions. Nous distinguons deux modes d'intégration implicite selon que l'unité des scénarios est préservée ou pas dans la spécification résultante de l'intégration. De manière informelle, nous supposons que l'unité des scénarios est préservée dans le modèle cible si les scénarios sont composés comme des blocs de Légo pour constituer la spécification qui les intègre. Ainsi, nous nous entendons d'appeler l'intégration préservant l'unité des scénarios par

intégration implicite Légo. Cependant, nous désignons par *intégration implicite libre*, l'autre type d'intégration implicite.

Les sections 7.2 et 7.3 sont respectivement consacrées à l'intégration implicite libre et à l'intégration implicite Légo de scénarios. A la section 7.4, nous combinons les deux modes d'intégration implicite précédents en un mode d'intégration implicite mixte de scénarios.

7.1 Matière première

Dans la section 5.1.4, nous avons défini trois ensembles H, V et Ev que nous associons à un système pour décrire ses données concernant les horloges, les variables discrètes et les événements. Puisque le comportement d'un système peut être décrit par plusieurs scénarios, nous désignons par S un ensemble de scénarios d'un système. Le quadruplet (H, V, Ev, S) représente la matière première pour la synthèse d'une spécification d'un système sous forme d'un automate temporisé. L'intégration des scénarios qui appartiennent à S permet de dériver un automate temporisé sécuritaire qui représente un prototype exécutable du système. Ainsi, lorsque nous parlons de l'exécution d'un système, il s'agit de la simulation du comportement de l'automate temporisé qui résulte de l'intégration des scénarios de S.

L'automate temporisé résultant de l'intégration des scénarios, doit être conforme à la sémantique des scénarios selon une relation de conformité qui caractérise le mode d'intégration. Cette relation de conformité permet de formaliser les propriétés des scénarios qui vont se retrouver dans la spécification résultant de leur intégration.

7.2 Intégration implicite libre

Nous qualifions d'implicite libre ce mode d'intégration, car durant l'exécution du système des comportements provenant de scénarios différents peuvent se succéder. L'ordre dans lequel ces comportements se succèdent est indirectement décrit par les scénarios. L'intégration implicite libre consiste à mettre toutes les actions des scénarios dans le même "*contenant*" que représente l'automate temporisé résultant de l'intégration. Ce '*contenant*" résulte en fait d'un ensemble d'actions-règles qui se compose des formes canoniques des scénarios à intégrer. L'automate qui résulte de l'intégration implicite libre est le ATS-EAR des actions-règles des formes canoniques respectives

des scénarios à intégrer.

Dans la suite de cette section nous formalisons le mode d'intégration implicite libre et nous formulons sa relation caractéristique de conformité.

7.2.1 Définition formelle de l'intégration implicite libre

Le mode d'intégration implicite libre est basé sur une granularité fine où l'unité élémentaire est l'action-règle. L'intégration implicite libre n'impose aucune restriction sur l'ordre d'exécution des actions-règles des scénarios. Après, l'exécution d'une action-règle, l'automate temporisé devrait permettre l'exécution de toute action-règle même si elle n'appartient pas au scénario qui est en cours d'exécution ; la seule condition est que l'état courant satisfasse les contraintes de l'action-règle.

Définition 7.1. *L'automate résultant de l'intégration implicite libre des scénarios de S est l'ATS-EAR qui est basé sur l'ensemble des actions-règles formé par l'union des formes canoniques des scénarios de S. Nous le notons $\mathcal{A}_S$.*

A partir de la définition précédente, nous déduisons un opérateur d'intégration implicite libre.

Définition 7.2. *Soit sc et sc' deux scénarios, nous définissons un opérateur d'intégration implicite libre sur les ATS-EARs respectifs des deux scénarios et nous le notons $\oplus$:*

$$A_{R(sc)} \oplus A_{R(sc')} \stackrel{def}{=} A_{R(sc) \cup R(sc')}$$

Proposition 7.1. *L'opérateur $\oplus$ est commutatif et associatif.*

$$A_{R(sc)} \oplus A_{R(sc')} = A_{R(sc')} \oplus A_{R(sc)}$$

$$(A_{R(sc)} \oplus A_{R(sc')}) \oplus A_{R(sc'')} = A_{R(sc)} \oplus (A_{R(sc')} \oplus A_{R(sc'')})$$

La proposition 7.1 permet de conclure que l'intégration implicite libre n'est pas sensible à l'ordre d'ajout des scénarios. Cette propriété est importante, elle est même nécessaire pour la construction incrémentale de la spécification d'un système par ajout progressif de scénarios.

7.2.2 Relation de conformité

La sémantique d'un scénario est donnée par son STE-EAR. Par conséquent, la relation de conformité qui caractérise le mode d'intégration implicite libre, se traduit par une relation de conformité entre les STE-EARs respectifs des scénarios et la spécification qui résulte de leur intégration. Le résultat de l'intégration est un automate temporisé sécuritaire dont la sémantique est donnée par le STE qui lui est associé. L'automate résultant permet tous les comportements des scénarios qu'il intègre. De manière plus spécifique, lorsque l'automate d'un scénario permet l'exécution d'une action-règle dans un état[1], l'automate où ce scénario est intégré le permet lui aussi dans un état similaire. La réciproque est fausse dans la mesure où l'automate, résultant de l'intégration de plusieurs scénarios, peut permettre l'exécution d'actions-règles que l'un des scénarios ne permet pas. Par conséquent, la relation de conformité qui relie l'automate d'un scénario et l'automate résultant de l'intégration de ce scénario avec d'autres, est une relation de simulation. D'où le théorème,

Théorème 7.1. *Pour chaque scénario $sc \in S$, il existe une relation de simulation entre le STE-EAR du scénario sc et le STE de $\mathcal{A}_S$.*

Démonstration. Le STE-EAR d'un scénario $sc \in S$ est noté $\Sigma_{R(sc)}$. Le LTS de $\mathcal{A}_S$ est noté $\Sigma_{\mathcal{A}_S}$. Pour montrer qu'il existe une simulation entre $\Sigma_{R(sc)}$ et $\Sigma_{\mathcal{A}_S}$, il suffit de construire une relation qui vérifie les points *(i a)* et *(ii a)* de la définition 2.6 page 13.
Soit la relation identité $I = \{(e, e') |$, e est un état de $\Sigma_{R(sc)}$, e' est un état de $\Sigma_{\mathcal{A}_S}$ et $e = e'\}$.
La relation I vérifie le point *(i a)* de la définition 2.6 car tout état de $\Sigma_{R(sc)}$ est un état de $\Sigma_{\mathcal{A}_S}$.
La relation I vérifie *(ii a)* de la définition 2.6 car pour tout état e, toute transition de $\Sigma_{R(sc)}$ qui part de e, est une transition de $\Sigma_{\mathcal{A}_S}$ et qui part aussi de e puisque les actions-règles de $R(sc)$ sont "présentes" dans $\Sigma_{\mathcal{A}_S}$. Ceci s'applique aussi pour les états d'arrivée de ces transitions.
D'où I est une relation de simulation entre $\Sigma_{R(sc)}$ et $\Sigma_{\mathcal{A}_S}$.

□

L'automate résultant de l'intégration —implicite libre— de plusieurs scénarios peut permettre certains comportements qui ne sont décrits dans aucun scénario mais qui résultent du chevauchement entre les scénarios. En d'autres termes, si nous considérons une trace dans l'automate d'intégration, il se peut que cette trace n'appartienne à aucun scénario mais qu'elle soit formée d'une concaténation de traces qui proviennent des scénarios intégrés. Par exemple, supposons que $l_1 l'_1$ soit une trace du scénario sc_1 et $l_2 l'_2$ de sc_2, il se peut que l'automate d'intégration de sc_1 et

[1] L'état est définie par un couple qui est composé d'une valuation de variables et une valuation d'horloge

sc_2 ait $l_1 l'_2$ comme une trace possible bien que $l_1 l'_2$ n'est ni trace de sc_1 ni de sc_2. La trace $l_1 l'_2$ résulte du chevauchement des scénarios sc_1 et sc_2. Une illustration du chevauchement entre les scénarios est schématisée à la figure 7.1.

L'intégration implicite libre des scénarios permet d'aboutir à une spécification où des comportements qui n'appartiennent à aucun scénario sont possibles. Cette anomalie, d'apparence seulement, est en fait la motivation du mode d'intégration implicite libre car elle permet de détecter toutes les interactions désirables ou non entre les services décrits par les scénarios. Souvent il s'agit d'interactions auxquelles le concepteur n'avait pas pensé. La spécification obtenue regroupe toutes les possibilités de combinaison des scénarios grâce à la fine granularité du mode d'intégration implicite libre. Cependant la spécification résultante du mode implicite libre n'est pas directement acceptée mais elle est soumise à une validation des comportements qui sont dûs aux chevauchements. Si l'un de ces comportements s'avère indésirable alors il révèle l'existence d'un oubli ou d'une mauvaise conception. Il suffit alors de corriger les scénarios qui sont reliés à ce comportement. La correction porte sur un ou plusieurs éléments de la description (V, H, Ev, S) du système.

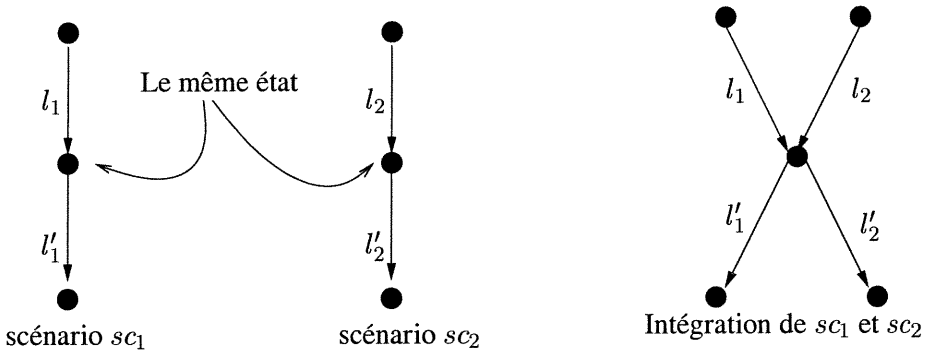


FIG. 7.1 – Schéma illustrant le chevauchement entre les scénarios

Le mode d'intégration implicite joue un rôle capitale dans la découverte des exigences des usagers, surtout dans la détection des interactions de services.

7.2.3 Exemples d'application

Nous allons traiter deux exemples d'application, l'un représente la spécification d'un commutateur téléphonique, l'autre traite un système de reconnaissance des différents types de clics d'une souris.

Exemple 1 : commutateur téléphonique

Nous reprenons l'exemple du commutateur téléphonique décrit à la section 6.3 en gardant la même description du domaine d'application donné par la figure 5.8 page 72. Nous supposons que le comportement du système est décrit par le scénario de la table 6.3 page 90 que nous appelerons scénario $sc1$. Ensuite nous étendons son comportement par un nouveau scénario que nous désignerons par $sc2$. Nous aboutissont à une specification du système sous forme d'un autoamate temporisé sécuritaire en utilisant le mode d'intégration implicite libre sur les deux scénarios $sc1$ et $sc2$. Étendant le comportement du système, le scénario $sc2$ décrit le cas où le poste A appelle le poste B et le trouve occupé. Le MSC de la figure 7.2 représente un schéma du scénario $sc2$, il indique que si le poste A appelle le poste B alors qu'il est occupé, le système envoie un $busy\_tone(A)$ au poste A.

a	φ_a	l_a	ψ_a	φ'_a	δ_a	λ_a
a_{00}	$True$	$pickup(A)$	$(A\_status{=}IDLE)$ $\wedge(A\_signal{=}NONE)$	$True$	$A\_status{:=}BUSY$	$\{h2\}$
a_{10}	$h2{=}0$	$send\_tone(A)$	$True$	$h2{=}0$	$A\_signal{:=}TONE$	$\{h1\}$
a_{20}	$h1{\leq}30$	$dial(A,B)$	$True$	$h1{<}30$	$A\_signal{:=}$ $DIALING(B)$	$\{h2\}$
a_{30}	$h2{=}0$	$busy\_tone(A)$	$(B\_status{=}BUSY)$	$h2{=}0$	$A\_signal{:=}$ $BUSY\_TONE$	$\{h2\}$

TAB. 7.1 – Description du scénario $sc2$ modélisant le comportement du commutateur lorsqu'un poste A appele un poste B alors qu'il est occupé

La modélisation du MSC de la figure 7.2 permet d'aboutir à la description de scénario $sc2$ donnée par à la table 7.1. Le résultat de l'intégration par le mode implicite libre des scénario $sc1$ et $sc2$, est l'ATS-EAR dont l'automate quotient est représenté par le graphe de la figure 7.3.

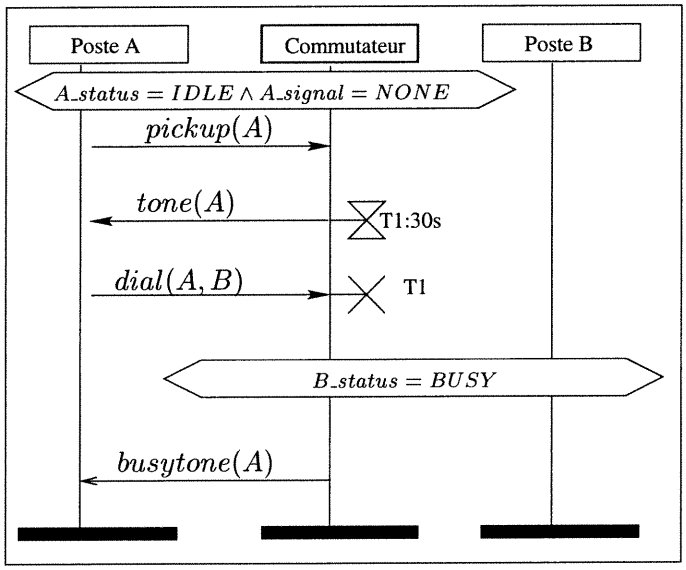


FIG. 7.2 – Un MSC qui schématise le scénario $sc2$

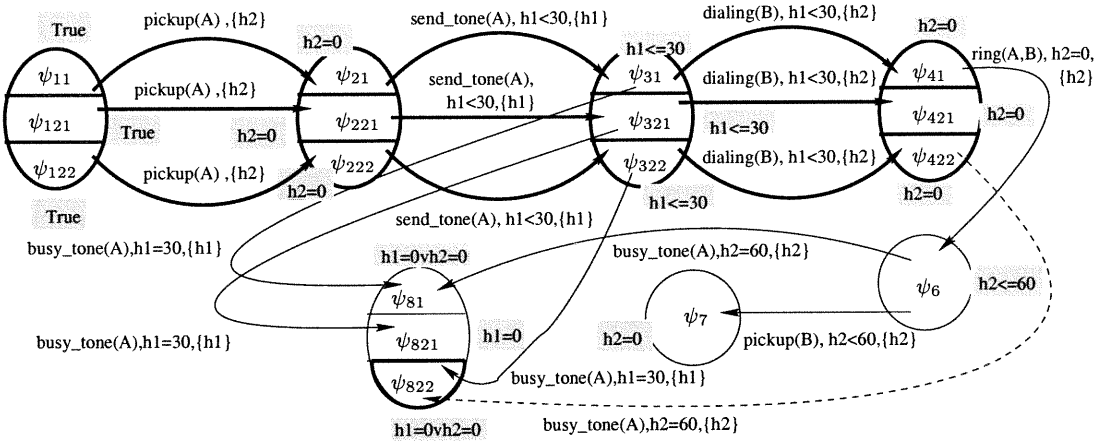


FIG. 7.3 – L'ATS-EAR quotient résultant de l'intégration implicite libre des scénarios $sc1$ représenté par la table 6.3 et $sc2$ représenté par la table 7.1

Seule l'action-règle de la dernière action principale (α_{80}) (table 7.1) du scénario $sc2$ n'est pas aussi une action-règle du scénario $sc1$. Donc l'extension du comportement du commutateur a consisté en l'ajout de cette action-règle. Cet ajout a conduit au partitionnement de la classe ψ_{42} de l'ATS-EAR quotient de sc_1 (figure 5.9) par la fonction $part$ en ψ_{421} et ψ_{422} lors du calcul de la partition initial. Ensuite l'application de la procédure $\omega\text{-}minimisation$ a occasionné le partitionnement des classes ψ_{32}, ψ_{22} et ψ_{12}, représentées dans la figure 5.9, en (ψ_{321} et ψ_{322}), (ψ_{221} et ψ_{222}) et (ψ_{121} et ψ_{122}) respectivement et représentées dans la figure 7.3.

Dans le graphe de la figure 7.3, nous avons représenté avec un trait continu fort toutes les transitions et les places de l'automate qui sont communes aux scénarios $sc1$ et $sc2$. Celles qui sont propres au scénario $sc1$ sont dessinées avec un trait continu fin. Il n'y a pas de place dans l'automate qui est propre au scénario $sc2$ par contre la transition dessinée avec un trait discontinu fin provient du scénario $sc2$ seulement. Le graphe de la figure 7.3 illustre l'intérêt du mode d'intégration implicite qui consiste à regrouper toutes les possibilités de combinaison des scénarios $sc1$ et $sc2$.

Exemple 2 : système de reconnaissance des clics d'une souris

Un système de reconnaissance de clics de souris, permet de générer les événements qui correspondent au clics simples, aux clics doubles et aux sélections lors de la manipulation d'une souris. Nous adaptons les exemples présentés dans [Sha92, Yov93] afin de constituer une application simple dont le comportement dépend du temps et qui permet d'illustrer l'intégration implicite des scénarios.

De manière informelle, un clic représente l'opération de presser le bouton de la souris puis de le relâcher avant un délai T_S. Nous décrivons le comportement correspondant à un clic par le scénario sc_1 qui est représenté par le MSC de la figure 7.4(a). L'événement du *clic simple* est observé si un clic est effectué puis suivi une durée de repos de T_{CD}. La détection du *clic simple* est modélisée par le scénario sc_2 à la figure 7.4(b). Un clic double est détecté si deux clics consécutifs sont séparés par un délai inférieur à T_{CD} ce qui est modélisé par le scénario sc_3 et représenté par la figure 7.4(c). Si le bouton de la souris a été pressé sans qu'il soit relâché avant le délai de T_S alors le système détecte le début de la sélection. La fin de la sélection est observée dès que le bouton est relâché. La sélection est traitée par le scénario sc_4 représenté à la figure 7.4(d). La constante T_S permet de faire la distinction entre les clics et la sélection. La constante T_{CD} représente le délai de

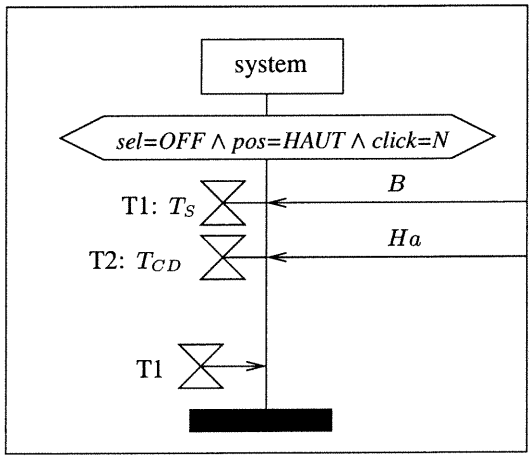
repos qu'il faut observer après un clic pour conclure que c'est un *clic simple*. Si ce repos de T_{CD} n'est pas respecté le premier clic peut faire partie d'un *clic double*.

La description du domaine représente la spécification des ensembles Ev, V et H (figure 7.5). L'ensemble des événements Ev du système comporte six alphabets qui sont CS marquant l'événement *clic simple*, CD pour l'événement *clic double*, DS indiquant le *début* de la *sélection* et FS pour la *fin* de la *sélection*. Les événements Ha et B indique le changement de la position du bouton de la souris vers le haut ou vers le bas. Notre modélisation du système à donné lieu à un ensemble V composé de trois variables discrètes qui sont *pos*, *click* et *sel*. La variable *pos* indique la position du bouton de la souris $HAUT$ ou BAS. La variable *click* mémorise si un clic a eu lieu et ses valeurs sont les constantes O pour oui et N pour non. Les valeurs de la variable *sel* sont ON et OFF indiquant si une opération de sélection est en cours ou pas.

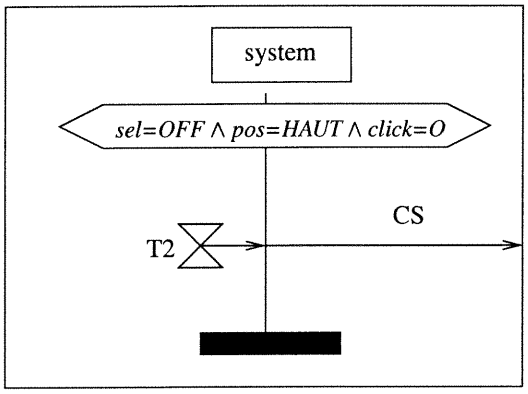
sc	a	Description de l' action $sc.a$					
		$\varphi_{sc.a}$	$l_{sc.a}$	$\psi_{sc.a}$	$\varphi'_{sc.a}$	$\delta_{sc.a}$	$\lambda_{sc.a}$
sc_1	a_{00}	$True$	B	$pos = HAUT \wedge click = N \wedge sel = OFF$	$True$	pos:=BAS	$\{h\}$
	a_{10}	$h \leq T_S$	Ha	$True$	$h < T_S$	pos:=$HAUT$, $click$:= O	$\{h\}$
sc_2	a_{00}	$h \leq T_{CD}$	CS	$pos = HAUT \wedge click = O \wedge sel = OFF$	$h = T_S$	$click$:=N	$\{h\}$
sc_3	a_{00}	$h \leq T_{CD}$	B	$pos = HAUT \wedge click = O \wedge sel = OFF$	$h < T_{DC}$	pos:=BAS	$\{h\}$
	a_{10}	$h \leq T_S$	Ha	$True$	$h < T_S$	pos:=$HAUT$	$\{h\}$
	a_{20}	$h = 0$	DC	$True$	$h = 0$	$click$:=N	$\emptyset$
sc_4	a_{00}	$True$	DS	pos=$BAS \wedge sel$=OFF	$h = T_S$	$click$:=N, sel:=ON	$\{h\}$
	a_{10}	$h \leq T_S$	Ha	$True$	$True$	pos:=$HAUT$	$\{h\}$
	a_{20}	$h = 0$	FS	$True$	$h = 0$	sel:=OFF	$\{h\}$

TAB. 7.2 – Description des scénarios sc_1, sc_2, sc_3 et sc_4 qui sont représentés par les MSCs de la figure 7.4

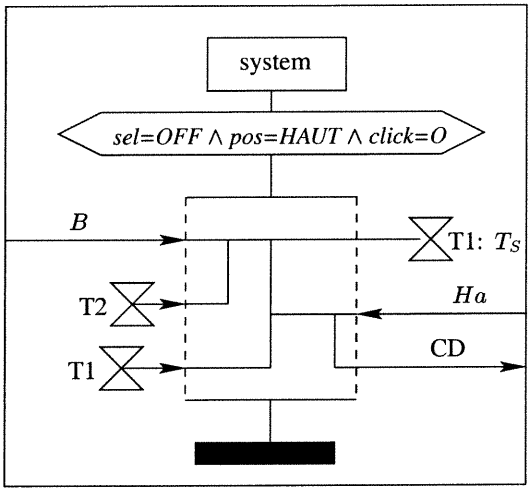
L'intégration des scénarios sc_1, sc_2, sc_3 et sc_4 de la table 7.2 donne lieu à l'automate temporisé sécuritaire représenté par la figure 7.6. Nous remarquons la place $(HAUT, O, OFF)$ est



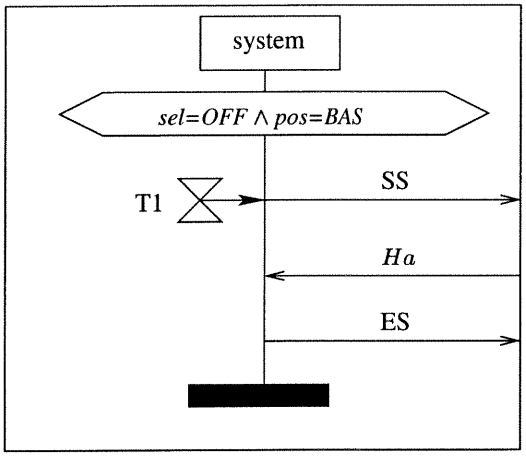
(a) sc_1 : scénario d'un *clic*



(b) sc_2 : scénario du *clic simple*



(c) sc_3 : scénario du *clic double*



(d) sc_4 : scénario de la *sélection*

FIG. 7.4 – Description du comportement du système de reconnaissance de clics sous forme de MSCs

$$Ev = \{CS, CD, DS, FS, Ha, B\}$$
$$V = \{pos, click, sel\}$$
$$dom(pos) = \{HAUT, BAS\}$$
$$dom(click) = \{O, N\}$$
$$dom(sel) = \{ON, OFF\}$$
$$H = \{h\}$$

FIG. 7.5 – Description du domaine d'application pour le système de reconnaissance des clics de la souris

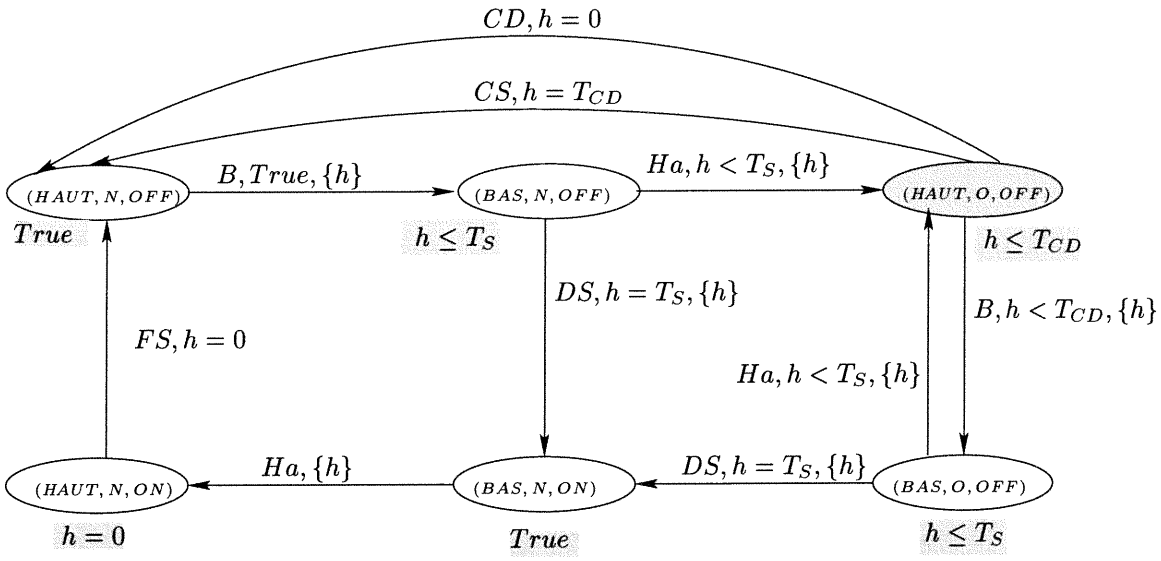


FIG. 7.6 – Graphe de l'automate temporisé sécuritaire obtenu par l'intégration implicite libre des scénarios sc_1, sc_2, sc_3 et sc_4 de la table 7.2. Chaque place de l'automate est caractérisée par un triplet de valeurs de la forme $(\omega(pos), \omega(click), \omega(sel))$ et une contrainte d'horloges comme invariant.

commune au scénario sc_1, sc_2 et sc_3. En analysant l'automate de la figure 7.6, nous trouvons que le chevauchement entre les scénarios sc_2 et sc_3 représente une erreur de conception car le système peut générer l'événement CD en exécutant la trace B,Ha,CD à partir de la place $(HAUT, N, OFF)$, ce qui est incorrect. Ce genre d'erreur, est une erreur type communément appelée *interaction de services*. Elle se produit lors du regroupement de services qui sont conçus séparément. L'intérêt du mode d'intégration implicite est la découvert de telles erreurs.

7.3 Intégration implicite Légo

Le mode d'intégration implicite Légo préserve l'unité d'un scénario dans la spécification résultante de l'intégration. Ainsi le système exécute un scénario du début jusqu'à sa fin avant de pouvoir commencer l'exécution d'un autre. Le mode d'intégration implicite Légo considère les scénarios comme non interruptibles. La fin d'un scénario est représentée par l'exécution d'une action qui mène à l'une des feuilles de son arbre. Cependant nous supposons que le début de l'exécution d'un scénario est marqué par l'exécution de l'une des actions qui appartiennent à son premier noeud principal.

Pour rencontrer les exigences du mode d'intégration implicite Légo, il faut être en mesure de distinguer les actions-règles d'un scénario à partir de leurs descriptions[2] pour éviter certains chevauchements. Ainsi nous allons enrichir les actions d'un scénario dans le but de garantir l'unité d'un scénario dans la spécification résultante de leur intégration.

Dans le reste de cette section, nous formalisons le mode d'intégration implicite Légo. Ensuite, nous identifions la relation de conformité caractéristique du mode d'intégration implicite Légo. Enfin, nous illustrons ce mode d'intégration par un exemple d'application.

7.3.1 Définition formelle de l'intégration implicite Légo

Dans l'intégration implicite Légo, lorsque l'exécution d'un scénario est commencée, il n'y a pas de chevauchement avec d'autres scénarios jusqu'à sa terminaison. Pour répondre à cette exigence, nous implantons un mécanisme d'exclusion mutuelle basé la notion de sémaphore. Il

[2]C'est pouvoir distinguer à quel scénario appartient une action-règle r rien qu'en considérant sa description $(\psi_r, \varphi_r, l_r, \varphi'_r, \delta_r, \lambda_r, \varphi''_r)$

s'agit d'ajouter une variable discrète spéciale que nous appelons *control*. La variable *control* est ajoutée à l'ensemble des variables V d'un système. Nous représentons $V' = V \cup \{control\}$. Nous supposons que chaque scénario de S a un identificateur unique. Les valeurs du domaine de la variable *control* sont les identificateurs des scénarios auxquels nous ajoutons une valeur spéciale désignée par NO_SC. Lorsque la variable *control* prend la valeur NO_SC, l'exécution de n'importe quel scénario dont les pré-conditions sont satisfaites, peut être entamée. Une fois l'exécution d'un scénario est commencée, la variable *control* prend comme valeur l'identificateur de ce scénario. Dès la terminaison de l'exécution d'un scénario, la variable *control* reprend la valeur NO_SC.

La normalisation d'un scénario consiste à l'enrichir pour en dériver un scénario où les opérations sur la variable *control* sont explicitement instanciées afin de garantir le comportement que nous venons de décrire. Nous définissons la forme normale d'un scénario par :

Définition 7.3. *Soient* $sc = (\mathcal{N}, \longmapsto)$ *un scénario et* sc_id *son identification unique. La forme normale du scénario* $sc \in S$ *est un scénario que nous notons* $\overline{sc}$*. Le scénario* $\overline{sc}$ *a la même structure que* sc *en terme du nombre de noeuds et du nombre d'actions par noeud. Ainsi, chaque action de* sc *admet un homologue dans* $\overline{sc}$*. Chaque action du scénario* $\overline{sc}$ *est obtenue à partir son action homologue dans le scénario* sc *par :*

- *L'action* $\overline{sc}.a_{00}$ *représente la première action principale*[3] *du scénario* $\overline{sc}$*. Elle est définie en fonction* $sc.a_{00}$ *par :*
  - $\psi_{\overline{sc}.a_{00}} \stackrel{def}{=} \psi_{sc.a_{00}} \wedge (control = NO\_SC)$
  - $\delta_{\overline{sc}.a_{00}} \stackrel{def}{=} \delta_{sc.a_{00}} \cup \{control := sc\_id\}$
  - $(\varphi_{\overline{sc}.a_{00}}, \varphi'_{\overline{sc}.a_{00}}, l_{\overline{sc}.a_{00}}, \lambda_{\overline{sc}.a_{00}}) \stackrel{def}{=} (\varphi_{sc.a_{00}}, \varphi'_{sc.a_{00}}, l_{sc.a_{00}}, \lambda_{sc.a_{00}})$
- *Pour toute action* $sc.a$ *telle que* $N \stackrel{sc.a}{\longmapsto} N'$*, où* $(N, N') \in \longmapsto$ *représente le couple de noeuds qui est associié à l'action* $sc.a$*, si le noeud* N' *est une feuille du scénario* sc *alors l'action homologue de* $sc.a$ *dans* $\overline{sc}$ *est définie par :*
  - $\delta_{\overline{sc}.a} \stackrel{def}{=} \delta_{sc.a} \cup \{control := NO\_SC\}$
  - $(\psi_{\overline{sc}.a}, \varphi_{\overline{sc}.a}, \varphi'_{\overline{sc}.a}, l_{\overline{sc}.a}, \lambda_{\overline{sc}.a}) \stackrel{def}{=} (\psi_{sc.a}, \varphi_{sc.a}, \varphi'_{sc.a}, l_{sc.a}, \lambda_{sc.a})$
- *Les autres actions de* $\overline{sc}$*, qui restent, sont exactement identiquesà leurs homologues respectifs dans* sc*.*

[3] cf. les définitions 6.1 et 6.2

Nous étendons la notation de la forme normale d'un scénario à un ensemble de scénarios. Ainsi $\overline{S}$ représente l'ensemble des formes normales respectives des scénarios appartenant à S.

L'intégration implicite Légo est caractérisée par une granularité où l'élément élémentaire est le scénario. En définissant la forme normale d'un scénario, nous réduisons le mode d'intégration implicite Légo à l'intégration implicite libre des formes normales respectives des scénarios à intégrer.

Définition 7.4. *L'automate résultant de l'intégration implicite libre des scénarios de S est l'ATS-EAR qui est basé sur l'ensemble des actions-règles qui est formé par l'union des formes canoniques des scénarios de $\overline{S}$. Nous le notons $\mathcal{A}_{\overline{S}}$.*

De même l'opérateur d'intégration implicite libre peut servir pour l'intégration implicite Légo de deux scénarios sc_1 et sc_2 en considérant l'automate :

$$A_{R(\overline{sc_1})} \oplus A_{R(\overline{sc_2})} = A_{R(\overline{sc_1}) \cup R(\overline{sc_2})}$$

7.3.2 Relation de conformité

Afin de définir la relation de conformité qui caractérise le mode d'intégration implicite Légo, nous formalisons la non interruption des exécutions des scénarios en définissant la notion de run.

Définition 7.5. *Nous appelons un "run" d'un scénario sc toute séquence temporisée observable finie $\sigma = e_0 \triangleright e_1 \triangleright e_2 \cdots \triangleright e_p$ de l'automate temporisé sécuritaire $A_{R(\overline{sc})}$ telle que*

- *e_0 satisfait $\overline{sc}.\psi_{N_0}$ et $\overline{sc}.\varphi_{N_0}$*
- *$\forall 0 < i < p$, e_i satisfait la contrainte $(control = sc\_id)$, où sc_id est l'identificateur du scénario sc.*
- *e_p satisfait $\overline{sc}.\psi_N$ et $\overline{sc}.\varphi_N$ où N est un noeud feuille[4] du scénario $\overline{sc}$*

La relation caractéristique du mode d'intégration implicite Légo peut être exprimée par le théorème suivant :

Théorème 7.2. *Soit $\sigma = e_1 \triangleright e_2 \triangleright e_3 \cdots \triangleright e_n$ une exécution observable de l'automate $\mathcal{A}_{\overline{S}}$ où S est un ensemble de scénarios. Nous supposons que σ vérifie les deux points suivants :*

[4]Les expressions de ψ_N et φ_N, où N est un noeud feuille d'un scénario, sont données par les équations 6.4 page 83

(1a) Il existe un scénario $sc \in S$ *tel que* e_1 *satisfait* $\overline{sc}.\psi_{N_0}$ *et* $\overline{sc}.\varphi_{N_0}$ *où* N_0 *est le premier noeud principal du scénario* $\overline{sc}$

(1b) Il existe un scénario $sc' \in S$ *tel que* e_n *satisfait* $\overline{sc'}.\psi_N$ *et* $\overline{sc'}.\varphi_N$ *où* N *est un noeud feuille de* $\overline{sc'}$.

Alors, il existe $sc_1, ..., sc_p \in S$, $e'_0, e'_1, e'_2, ..., e'_{p-1}, e'_p$ *des états du STE de* $\mathcal{A}_{\overline{S}}$ *et* $\sigma_1, \sigma_2, ..., \sigma_p$ *des exécutions observables de* $\mathcal{A}_{\overline{S}}$ *tels que :*

(2a) σ_i *est un run du scénario* sc_i *pour tout* $1 \leq i \leq p$

(2b) et $\sigma = e'_0 \triangleright \sigma_1 \triangleright e'_1 \triangleright \sigma_2 \triangleright e'_2 \triangleright \sigma_3 \triangleright e'_3 \triangleright ... \triangleright e'_{p-1} \triangleright \sigma_p \triangleright e'_p$

Le théorème 7.2 exprime qu'en satisfaisant les point $(1a)$ et $(1b)$, toute exécution d'un automate qui intègre un ensemble de scénarios selon la méthode d'intégration implicite Légo, est en fait une séquence de run de scénarios éventuellement séparés par des écoulements du temps comme c'est décrit par les points $(2a)$ et $(2b)$. Les conditions des points $(1a)$ et $(1b)$ permettent d'assurer que le début et la fin de l'exécution σ sont alignés sur le début et la fin de certains scénarios.

Pour montrer le théorème 7.2, nous commençons par montrer la proposition suivante :

Proposition 7.2. *Si une exécution observable* σ *vérifie les points* $(1a)$ *et* $(1b)$ *du théorème 7.2 alors :*

(i) soit il existe un scénario sc_1, *un run* σ_1 *de* $\overline{sc_1}$, *un état* e' *du STE de* $\mathcal{A}_{\overline{S}}$ *et* σ' *une exécution observable de* $\mathcal{A}_{\overline{S}}$ *tels que* $\sigma = e' \triangleright \sigma_1 \triangleright \sigma'$ *et* σ' *vérifie les points* $(1a)$ *et* $(1b)$ *du théorème 7.2*

(ii) soit il existe un scénario sc_1, *un run* σ_1 *de* $\overline{sc}$ *et deux état* e' *et* e'' *du STE de* $\mathcal{A}_{\overline{S}}$ *tels que* $\sigma = e' \triangleright \sigma_1 \triangleright e''$.

Démonstration. La preuve de cette proposition est basée sur l'utilisation des règles de réécriture des séquences d'exécution d'un automate temporisé.
Étant donnée $\sigma = e_1 \triangleright e_2 \triangleright e_3 \cdots \triangleright e_n$ une exécution observable fini de $\mathcal{A}_{\overline{S}}$, par hypothèse selon le point $(1a)$, il existe un scénario sc tel que e_1 satisfait $\overline{sc}.\varphi_{N_0}$ et $\overline{sc}.\psi_{N_0}$ où N_0 est le premier noeud principal du scénario $\overline{sc}$.
Puisque σ est un exécution observable, il existe un pas observable dans σ. Soi $i < n$ le plus petit entier tel que $e_i \triangleright e_{i+1}$ est un pas observable.
$\forall 1 \leq j < i$, $\exists d_j \in \mathbb{R}^+$ tel que $e_j \triangleright e_{j+1}$ représente $e_j \xrightarrow{d_j} e_j + d_j \xrightarrow{0} e_{j+1}$
$\forall 1 \leq j < i$, les transitions $e_j + d_j \xrightarrow{0} e_{j+1}$ bouclent sur le même état.
Le pas $e_i \triangleright e_{i+1}$ représente $e_i \xrightarrow{d_i} e_i + d_i \xrightarrow{l_i} e_{i+1}$ où $l_i \in Ev$ est une étiquette observable.

Ainsi, la séquence de pas de taille i qui préfixe σ peut être réduite par :

$$e_1 \triangleright e_2 \triangleright e_3 \triangleright \cdots \triangleright e_i \triangleright e_{i+1} = e_1 \triangleright e_i \triangleright e_{i+1} \tag{7.1}$$

Nous savons que e_i satisfait la contrainte $(control = NO\_SC)$ car e_i satisfait $(\overline{sc}.\psi_{N_0})$.
Soit $i' > i$ le plus petit entier tel que $e_{i'}$ satisfait la contrainte $(control = NO\_SC)$. i' existe bien car e_n satisfait la contrainte $(control = NO\_SC)$.
Nous affirmons qu'il existe un scénario sc_1 tel que la séquence :

$$\sigma_1 = e_i \triangleright e_{i+1} \triangleright \cdots \triangleright e_{i'} \tag{7.2}$$

est un run de $\overline{sc_1}$. En effet, il suffit de considérer que sc_1 est le scénario d'où provient la transition $e_i + d_i \xrightarrow{l_i} e_{i+1}$. Ainsi nous pouvons écrire,

$$\sigma = e_1 \triangleright e_i \triangleright e_{i+1} \triangleright \ldots \triangleright e_{i'} \triangleright e_{i'} \triangleright e_{i'+1} \triangleright \cdots \triangleright e_n \tag{7.3}$$

$$= e_1 \triangleright \sigma_1 \triangleright e_{i'} \triangleright e_{i'+1} \triangleright \cdots \triangleright e_n \tag{7.4}$$

En considérant la séquence de pas suivante

$$e_{i'} \triangleright e_{i'+1} \triangleright \cdots \triangleright e_n \tag{7.5}$$

nous distinguons deux cas selon que la séquence (7.5) est observable ou pas.
Dans le cas où la séquence (7.5) n'est pas observable, alors elle ne comporte que des transitions d'écoulement du temps qui peuvent être regroupées en utilisant l'additivité du temps. Nous obtenons alors,

$$\sigma = e_1 \triangleright \sigma_1 \triangleright e_n \tag{7.6}$$

L'équation (7.6) coïncide avec le point (i) de la proposition (7.2).

L'autre cas correspond à la situation où la séquence (7.5) est observable. Soit $i'' \geq i'$ le premier entier tel que le pas $e_{i''} \triangleright e_{i''+1}$ est observable dans la séquence (7.5). Il existe alors une durée $d_{i''}$ et un événement $l_{i''} \neq 0$ tels que $e_{i''} \triangleright e_{i''+1}$ représente $e_{i''} \xrightarrow{d_{i''}} e_{i''} + d_{i''} \xrightarrow{l_{i''}} e_{i''+1}$.

Par conséquent il existe un scénario sc_2 tel que l'état $e_{i''} + d_{i''}$ satisfait $\overline{sc_2}.\varphi_{N_0}$ et $\overline{sc_2}.\psi_{N_0}$.
Nous pouvons déduire que la séquence :

$$\sigma' = (e_{i''} + d_{i''}) \triangleright e_{i''+1} \triangleright \cdots \triangleright e_n \tag{7.7}$$

vérifie les points $(1a)$ et $(1b)$ de le théorème 7.2. Par contre, en regroupant tous les pas non observables qui sont entre $e_{i'}$ et $e_{i''}$, nous obtenons

$$e_{i'} \triangleright e_{i'+1} \triangleright \cdots \triangleright e_{i''} \triangleright \cdots \triangleright e_n = e_{i'} \triangleright (e_{i''} + d_{i''}) \triangleright e_{i''+1} \triangleright \cdots \triangleright e_n \tag{7.8}$$

En récapitulant les résultats, nous avons

$$\begin{aligned} \sigma &= e_1 \triangleright \sigma_1 \triangleright e_{i'} \triangleright (e_{i''} + d_{i''}) \triangleright e_{i''+1} \triangleright \cdots \triangleright e_n && (7.9) \\ &= e_1 \triangleright \sigma_1 \triangleright e_{i'} \triangleright \sigma' && (7.10) \end{aligned}$$

Puisque le pas $\triangleright e_{i'}$ dans (7.10) boucle sur le même état car $e_{i'}$ est le dernier état dans la séquence σ_1 (7.2), nous pouvons écrire

$$\sigma = e_1 \triangleright \sigma_1 \triangleright \sigma' \tag{7.11}$$

L'équation (7.11) coïncide avec le point (ii) de la proposition (7.2).

□

Une preuve du théorème 7.2 est déduite en appliquant la proposition 7.2. Puisque le nombre de pas de σ dans le théorème 7.2 est fini, nous appliquons la proposition 7.2 un nombre de fois au plus égal au nombre de pas dans σ pour montrer le théorème 7.2.

Remarque 7.1. *Etant donné un scénario $sc \in S$, les automates $A_{R(sc)}$ et $\mathcal{A}_{\overline{S}}$ ne sont pas forcément similaires à l'encontre de $A_{R(sc)}$ $\mathcal{A}_S$ qui le sont d'après le théorème 7.1.*

La sémantique du mode d'intégration Légo est favorisée lorsque les scénarios modélisent des services et qu'un service doit être complété sans interruption avant de commencer un autre.

7.3.3 Exemple d'application

Nous reprenons l'exemple d'application traitant le système de reconnaissance des clics d'une souris, mais cette fois pour illustrer le mode d'intégration implicite Légo des scénarios. Nous allons aussi comparer les résultats obtenus lors de l'intégration implicite libre avec ceux du mode implicite Légo.

Nous supposons ici que le système de reconnaissance des clics d'une souris respecte les exigences décrites dans le deuxième exemple d'application de la section 7.2.3 par la description du domaine d'application représenté par la figure 7.5 page 102 et seulement les spécifications des scénarios sc_1, sc_2 et sc_3 données par la table 7.2 page 100.

Nous rappelons que l'intégration des scénarios sc_1, sc_2 et sc_3 par le mode implicite Légo revient à intégrer les scénarios $\overline{sc_1}$, $\overline{sc_2}$ et $\overline{sc_3}$ avec le mode implicite libre. L'automate résultant de cette intégration est représenté par le graphe de la figure 7.7. Nous remarquons qu'il y a plus d'interactions de services entre les scénarios sc_2 et sc_3 comme c'était le cas dans l'automate résultant l'intégration du mode implicite repéssenté par la figure 7.6 page 102. Ceci est dû à la sémantique du mode implicite Légo qui permet protéger un scénario contre les chevauchements.

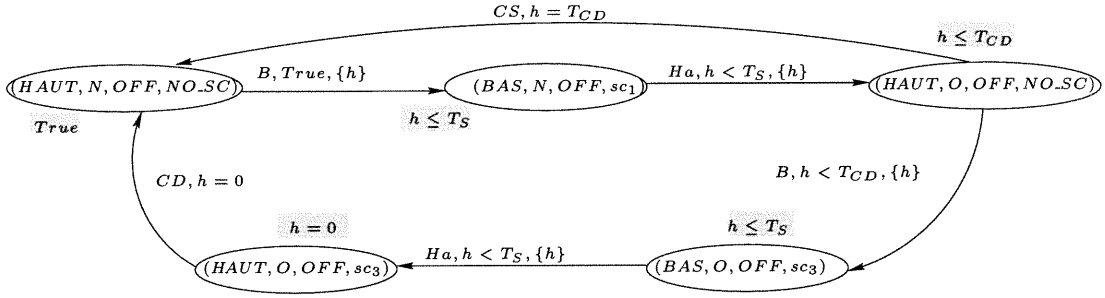


FIG. 7.7 – Graphe de l'automate temporisé sécuritaire obtenu par l'intégration implicite Légo des scénarios sc_1, sc_2 et sc_3 de la table 7.2. Chaque place de l'automate est caractérisée par une contrainte d'horloges comme invariant et un quadruplet de valeurs de la forme $(\omega(pos), \omega(click), \omega(sel), \omega(control)) \in \Omega(V \cup \{control\})$.

7.4 Intégration implicite mixte

Le mode d'intégration mixte est le fruit de la combinaison des deux modes d'intégration implicite libre et implicite Légo dans la même spécification. Il revient au concepteur de spécifier pour chaque scénario le mode d'intégration qui peut être soit l'implicite libre ou bien l'implicite Légo. Le choix de l'un ou l'autre des modes implicites dépend de la sémantique que le concepteur veut attribuer au comportement que le scénario véhicule. Si le concepteur veut que l'exécution d'un scénario soit non interruptible, il peut choisir le mode d'intégration implicite Légo pour ce

scénario. Autrement, si le concepteur veut découvrir toutes les interactions possibles d'un scénario avec la spécification courante[5], il peut choisir le mode d'intégration implicite libre pour intégrer ce scénario.

Dans les deux sections précédentes, nous avons défini les deux modes d'intégration implicites que nous qualifions de purs car dans chacun des deux cas, tous les scénarios sont intégrés par le même mode d'intégration. En combinant les ces deux modes d'intégration implicite Légo et implicite libre dans la même spécification, le problème majeur est d'assurer que la sémantique de chacun des modes est préservée et qu'il n'y a pas d'interaction de sémantique entre les deux modes.

Un autre problème technique concerne l'ensemble des variables discrètes des scénarios. Dans le mode implicite Légo l'ensemble des variables discrètes est $V' = V \cup \{control\}$ alors que l'ensemble des variables discrètes dans le mode d'intégration implicite libre est V.

Dans la suite de cette section, nous définissons la combinaison des deux modes d'intégration implicites en un mode d'intégration implicite mixte. Ensuite, nous comparons la relation de conformité du mode d'intégration implicite mixte par rapport aux relations de conformité des modes d'intégration implicites purs qui sont les modes libre et Légo.

7.4.1 Définition du mode d'intégration implicite mixte

Dans le mode d'intégration implicite mixte, il faut spécifier pour chaque scénario le mode d'intégration à utiliser. Lorsque le mode d'intégration implicite Légo est choisi, c'est la forme canonique du scénario normalisé qui sera intégrée. Par contre, dans le cas où le mode de l'intégration implicite libre est choisi, c'est la forme canonique du scénario qui sera intégrée. La variable $control$ est instanciée seulement dans le cas des scénarios normalisés. Nous rappelons que $control$ est une variable spéciale que nous avons introduite pour permettre le mode d'intégration Légo.

Le fait que la forme canonique d'un scénario non normalisé n'instancie jamais la variable $control$ risque d'induire une explosion combinatoire inutile du nombre de places dans l'automate temporisé sécuritaire de la spécification. De plus, il risque aussi de nuire à la sémantique du mode

[5] représente les scénarios qui sont déjà intégrés

d'intégration implicite Légo dans le mode mixte. En effet, aucune contrainte sur variable ne se rapporte à la variable *control* dans la forme canonique d'un scénario, ainsi toute valeur qui appartient au domaine de la variable *control* est possible. En particulier, la variable *control* peut prendre comme valeur l'identité d'un scénario qui est déjà intégré à la spécification selon le mode implicite Légo. Par conséquent, l'exécution de ce scénario risque d'être interrompu à cause de son chevauchement avec d'autres scénarios qui sont intégrés par le mode implicite libre.

Pour résoudre ce problème, nous allons explicitement instancier la variable *control* dans les actions-règles des scénarios où elle ne l'est pas. Seules les actions-règles des scénarios qui sont intégrés en implicite libre seront touchées par cette opération. Ainsi, l'exécution d'un scénario qui est intégré selon le mode implicite libre ne pourra jamais interagir avec l'exécution d'un scénario qui est intégré par le mode implicite Légo.

Définition 7.6. *Soit $S' \subset S \cup \overline{S}$ où S est un ensemble de scénarios. Nous considérons les ensembles d'actions-règles suivants :*
$R = \{(\psi_r \wedge (control = NO\_SC), \varphi_r, l_r, \phi_r, \delta_r, \lambda_r, \varphi'_r) \mid \exists r \in R(sc) \,.\, sc \in S' \cap S\}$
$R' = \{r \in R(\overline{sc}) \mid \exists sc \in S \,.\, \overline{sc} \in S'\}$
L'automate résultant de l'intégration implicite mixte des scénarios de S' est l'ATS-EAR qui est basé sur l'ensemble des actions-règles $R \cup R'$. Nous le notons $\mathcal{A}_{S'}$.

Un exemple de l'ensemble S' de la définition 7.6 est $S' = \{sc_1, sc_2, \overline{sc_3}\}$ où les scénarios sc_1, sc_2 et sc_3 sont des scénarios qui appartiennent à S. L'intégration implicite mixte des scénarios de S' consiste à intégrer les scénarios sc_1 et sc_2 par le mode implicite libre et le scénario sc_3 selon le mode implicite Légo.

Dans la définition 7.6, R est un ensemble d'actions-règles dont les éléments sont obtenus en modifiant les actions-règles provenant de scénarios qui doivent être intégrés selon le mode implicite libre. Cette modification consiste à sur-contraindre chaque action-règle r en remplaçant ψ_r par $\psi_r \wedge (control = NO\_SC)$. Ainsi nous évitons tout chevauchement non désiré entre les scénarios du mode implicite libre et ceux du mode implicite Légo. Par ailleurs, R' représente l'ensemble des actions-règles qui proviennent des scénarios qui doivent être intégrés selon le mode implicite Légo. Dans ce cas, nous considérons les actions-règles des scénarios normalisés.

7.4.2 Relation de conformité

Dans cette section nous allons montrer que les relations de conformité des modes d'intégration implicite Légo et implicite libre sont préservées lorsque ces deux modes sont combinés.

Nous allons voir que la relation de conformité caractéristique du mode d'intégration implicite libre (théorème 7.1) est encore vérifié lors de l'intégration implicite mixte.

Proposition 7.3. *Soit $S' \subset S \cup \overline{S}$ où S est un ensemble de scénarios. Pour chaque scénario $sc \in S' \cap S$, il existe une relation de simulation entre le STE-EAR du scénario sc et le STE de $\mathcal{A}_{S'}$.*

Démonstration. La preuve de cette proposition est similaire à celle du théorème 7.1. Elle consiste à construire une relation de simulation entre les STEs $\Sigma_{R(sc)}$ et $\Sigma_{\mathcal{A}_{S'}}$ telle que $sc \in S'$. Les états du LTS $\Sigma_{R(sc)}$ sont des éléments qui appartiennent à $\Omega(V) \times \Theta(H)$, par contre les états de $\Sigma_{\mathcal{A}_{S'}}$ sont des éléments de $\Omega(V') \times \Theta(H)$ où $V' = V \cup \{control\}$.

Nous désignons par $proj_{control}$ l'application qui associe à une assignation de variables de $\Omega(V')$ sa projection sur $\Omega(V)$ par rapport à la dimension de la variable $control$. En posant, $\omega = proj_{control}(\omega')$ où $\omega' \in \Omega(V')$, Nous définissons $\omega \in \Omega(V)$ par $\forall v \in V, \omega(v) \stackrel{=}{def} \omega'(v)$.
Soit la relation $P_{control}$ définie par :

$$\begin{aligned} P_{control} \quad = \quad & \{(e, e') \mid e = (\omega, \theta) \text{ est un état de } \Sigma_{R(sc)}, \\ & e' = (\omega', \theta') \text{ est un état de } \Sigma_{\mathcal{A}_{S'}} \ . \\ & \theta' = \theta \text{ et } proj_{control}(\omega') = \omega\} \end{aligned} \tag{7.12}$$

La relation $P_{control}$ est une relation de simulation entre $\Sigma_{R(sc)}$ et $\Sigma_{\mathcal{A}_{S'}}$.

□

Dans le mode implicite mixte, l'exécution d'un scénario, intégré selon le mode implicite Légo, est non interruptible comme le montre la proposition suivante :

Proposition 7.4. *Soit $S' \subset S \cup \overline{S}$. Soit $\sigma = e_1 \triangleright e_2 \triangleright e_3 \cdots \triangleright e_n$ une exécution observable de l'automate $\mathcal{A}_{S'}$. Nous supposons qu'il existe un scénario normalisé $\overline{sc} \in S' \cap \overline{S}$ tel que σ vérifie les deux points suivants :*

(1a) $e_1 \triangleright e_2$ est un pas observable qui provient du scénario $\overline{sc}$ tel que e_1 satisfait les contraintes $\overline{sc}.\psi_{N_0}$ et $\overline{sc}.\varphi_{N_0}$ dans lesquelles N_0 est le premier noeud principal du scénario $\overline{sc}$

(1b) e_n satisfait les contraintes $\overline{sc}.\psi_N$ et $\overline{sc}.\varphi_N$ dans lesquelles N est un noeud feuille de $\overline{sc}$.

Alors il existe un entier $p \leq n$ tel que $e_1 \triangleright e_2 \triangleright e_3 \triangleright \cdots \triangleright e_p$ est un run du scénario sc.

Démonstration. Nous proposons une démonstration de la proposition 7.4 en utilisant les mêmes techniques que la preuve de la proposition 7.2.
Le point $(1a)$ montre que le scénario sc qui est intégré selon le mode implicite Légo et que le pas observable $e_1 \triangleright e_2$ marque le début de l'exécution du scénario $\overline{sc}$.
Le point $(1b)$ exprime le fait que l'état e_n satisfait la contraint $(control = NO\_SC)$ car N est un noeud feuille du scénario $\overline{sc}$. Par conséquent l'exécution du scénario sc, qui a commencé à l'état e_1, s'est déjà terminée.
Le pas $e_1 \triangleright e_2$ représente $e_1 \xrightarrow{d_1} e_1 + d_1 \xrightarrow{l_1} e_2$ où d_1 est un nombre réel et $l_1 \neq 0$ appartient à Ev.
Nous distinguons deux cas selon que e_2 satisfait ou pas la contrainte $(control = sc\_id)$:

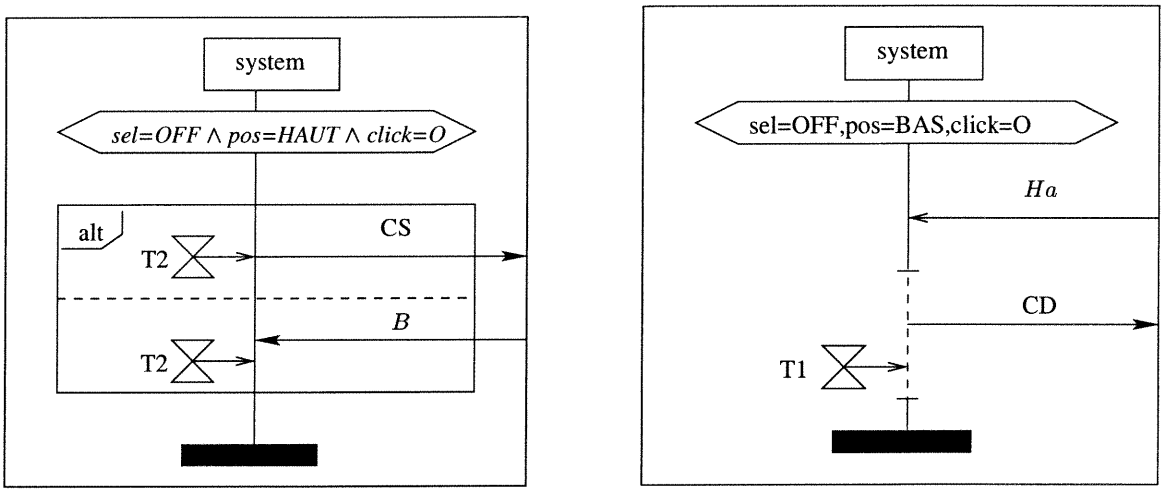
- Cas où e_2 ne satisfait pas la contrainte $(control = sc\_id)$: e_2 satisfait la contrainte $(control = NO\_SC)$. Le pas $e_1 \triangleright e_2$ correspond à l'exécution d'une action secondaire qui appartient au premier noeud principal N_0 du scénario $\overline{sc}$ ou bien à l'exécution de la première action principale si le scénario $\overline{sc}$ ne comporte que deux noeuds principaux. Par conséquent, $e_1 \triangleright e_2$ est un run du scénario sc.
- Cas où e_2 satisfait la contrainte $(control = sc\_id)$: puisque e_n satisfait la contrainte $(control = NO\_SC)$, soit $p \leq n$ le premier entier tel que e_p satisfait la contrainte $(control = NO\_SC)$. Tous les état e_i tels que $1 < i < p$, satisfont la contrainte $(control = sc\_id)$ car pour que la variable $control$ prend une autre valeur il faut qu'elle passe par NO_SC mais e_p représente le premier état après e_1 où la contrainte $(control = NO\_SC)$ est satisfaite. Donc $e_1 \triangleright e_2 \triangleright e_3 \triangleright \cdots \triangleright e_p$ est un run du scénario sc.

□

Les propositions 7.3 et 7.4 montrent le mode implicite mixte préserve la sémantique de chacun des deux modes implicite libre et implicite Légo lors de leur combinaison. Le concepteur pourra utiliser le mode d'intégration implicite mixte et décider des scénarios qui seront intégrés en implicite Libre et ceux qui le seront en implicite Légo sans craindre d'interaction de sémantique. Cette possibilité permet de faciliter l'utilisation de notre approche.

7.4.3 Exemples d'application

Pour illustrer le mode d'intégration mixte de scénarios, nous considérons une nouvelle spécification du système de reconnaissance des clics d'une souris. Nous adoptons la description du domaine d'application donnée à la figure 7.5 et le scénario sc_4 qui est décrit par la table 7.2 et par le MSC de la figure 7.4 (d). Nous supposons aussi que le comportement du système suit les scénarios sc'_1, sc'_2 et sc'_3 qui sont spécifiés par la table 7.3. Une description sous forme de MSCs est fournie à la figure 7.8 pour sc'_2 et sc'_3. Bien que les scénarios sc'_2 et sc'_3 soit deux nouveaux scénarios, ils sont seulement taillés différemment des scénarios sc_2 et sc_3 de la table 7.2 et permettent le même comportement. Nous exprimons cela de manière formelle par $R(sc_2) \cup R(sc_3) = R(sc'_2) \cup R(sc'_3)$. Par contre le scénario sc'_1 montre une différence concernant les contraintes d'horloges de l'action $sc'_1.a_{00}$ par rapport l'action $sc_1 a_{00}$ du scénario sc_1 de la table 7.2.



(a) Scénario sc'_2 distingant un *clic simple* du *double*

(b) Scénario sc'_3 détectant d'un *clic double*

FIG. 7.8 – Description du comportement des nouveaux scénarios sc'_2 et sc'_3 sous forme de MSCs

Dans cet exemple d'application nous comparons les automates temporisés résultant de l'intégration des scénarios selon les modes mixte, implicite libre et implicite Légo. Pour l'intégration mixte nous considérons l'ensemble de scénarios $\{sc'_1, sc'_2, \overline{sc'_3}, sc_4\}$ pour lequel l'automate temporisé résultant est représenté à la figure 7.9. Le mode d'intégration implicite libre des scénarios

sc	a	Description de l' action $sc.a$					
		$\varphi_{sc.a}$	$l_{sc.a}$	$\psi_{sc.a}$	$\varphi'_{sc.a}$	$\delta_{sc.a}$	$\lambda_{sc.a}$
sc'_1	a_{00}	$True$	B	$pos = HAUT \wedge click = N \wedge sel = OFF$	$h < T_{CD}$	pos:=BAS	$\{h\}$
	a_{10}	$h \leq T_S$	Ha	$True$	$h < T_S$	pos:=$HAUT$, $click$:= O	$\{h\}$
sc'_2	a_{00}	$h \leq T_{CD}$	CS	$pos = HAUT \wedge click = O \wedge sel = OFF$	$h = T_{CD}$	$click$:=N	$\{h\}$
	a_{01}	$h \leq T_{CD}$	B	$True$	$h < T_{DC}$	pos:=BAS	$\{h\}$
sc'_3	a_{00}	$h \leq T_S$	Ha	$pos = BAS \wedge click = O \wedge sel = OFF$	$h < T_S$	pos:=$HAUT$	$\{h\}$
	a_{10}	$h = 0$	DC	$True$	$h = 0$	$click$:=N	$\emptyset$

TAB. 7.3 – Spécification des nouveaux scénarios sc'_1, sc'_2 et sc'_3

est obtenu en appliquant le mode d'intégration mixte sur l'ensemble $\{sc'_1, sc'_2, sc'_3, sc_4\}$ pour lequel l'automate temporisé résultant de l'intégration ressemble à l'automate représenté à la figure 7.6. Le mode d'intégration Légo consiste à pratiquer le mode d'intégration mixte sur l'ensemble de scénarios $\{\overline{sc'_1}, \overline{sc'_2}, \overline{sc'_3}, \overline{sc_4}\}$. Le graphe de l'automate temporisé résultant dans ce cas est représenté à la figure 7.10.

L'automate de la figure 7.10 résultant du mode implicite Légo, souffre de non déterminisme aux places $(HAUT, N, OFF, NO\_SC)$ et $(HAUT, N, ON, NO\_SC)$. Cependant, nous avons une interaction de service entre sc_2 et sc_3 dans l'automate résultant du mode implicite libre. Grâce au mode implicite mixte, nous avons obtenu l'automate de la figure 7.10 qui ne présente pas les anomalies des deux premiers.

7.5 Conclusion

Dans ce chapitre nous avons proposé deux modes d'intégration des scénarios que nous qualifions d'implicites et purs. Il s'agit des modes d'intégration implicite libre et l'implicite Légo. Le mode d'intégration implicite libre se distingue par sa fine granularité qui est basée sur l'action d'un scénario. Ce mode est utile pour la découverte des exigences des utilisateurs ainsi que pour repérer les interactions possibles de services.

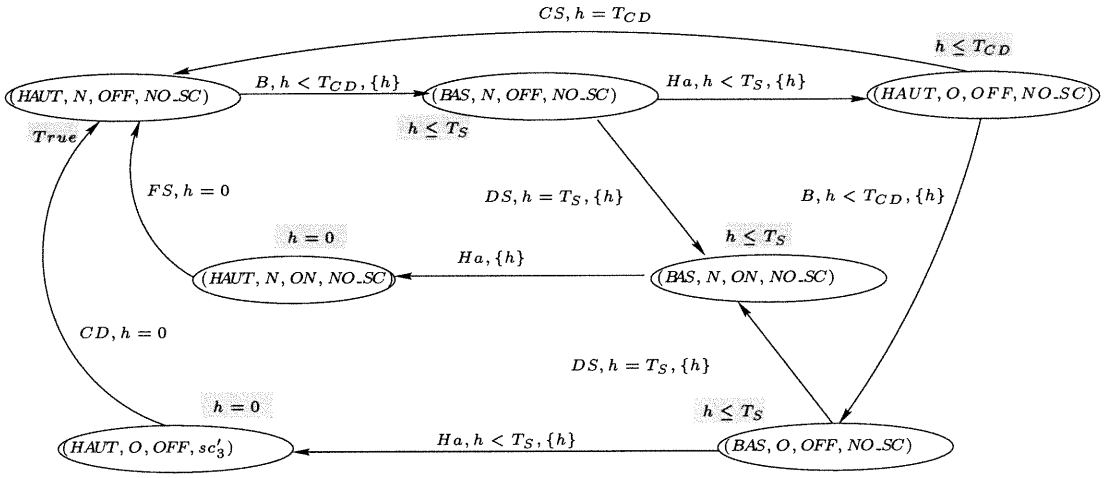


FIG. 7.9 – Graphe de l'automate temporisé sécuritaire obtenu par l'intégration implicite mixte de l'ensemble des scénarios $\{sc'_1, sc'_2, \overline{sc'_3}, sc_4\}$

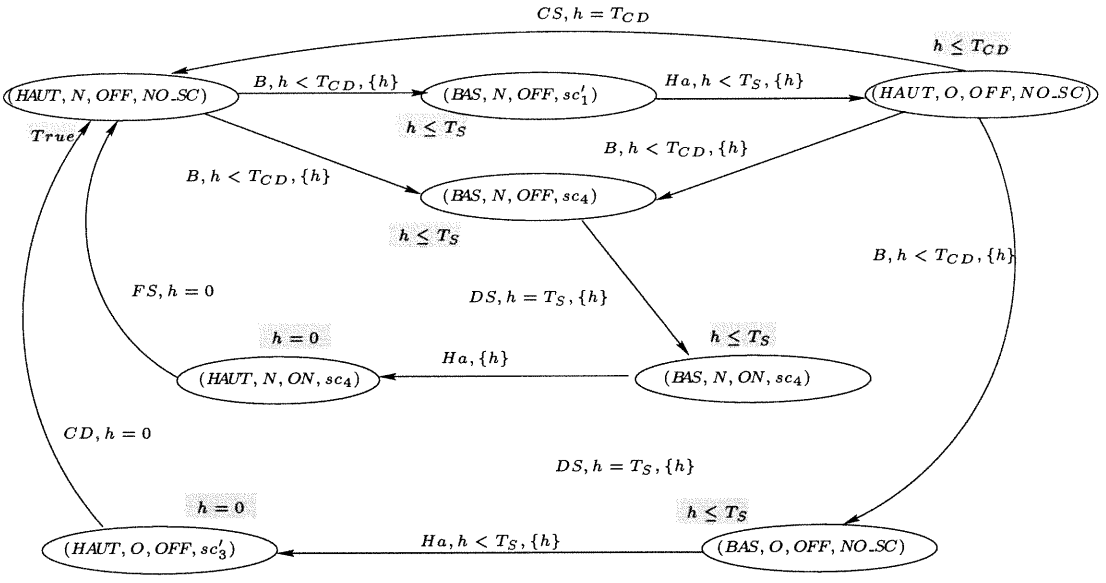


FIG. 7.10 – Graphe de l'automate temporisé sécuritaire obtenu par l'intégration implicite Légo des scénarios $\overline{sc'_1}$, $\overline{sc'_2}$, $\overline{sc'_3}$ et $\overline{sc_4}$

Le mode d'intégration implicite Légo permet de protéger l'unité de chaque scénario dans la spécification générée. Dans ce mode, les scénarios représentent les blocs qui composent la spécification. L'exécution d'un scénario n'est pas interruptible car la granularité du mode d'intégration implicite Légo est le scénario.

Le mode d'intégration implicite mixte offre la possibilité de combiner les modes d'intégration implicite libre et Légo dans la même spécification. De plus le mode mixte préserve la sémantique de chacun des deux modes implicites combinés permettant ainsi aux concepteurs une facilité d'utilisation. Dans le mode d'intégration implicite mixte, les concepteurs peuvent choisir pour chaque scénario le mode d'intégration implicite à utiliser libre ou Légo, selon la spécification qu'il veulent véhiculer.

Chapitre 8

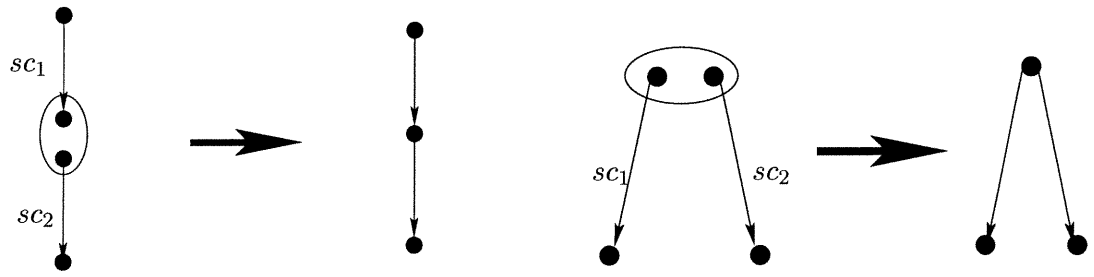
Intégration explicite des scénarios temporisés

L'intégration explicite des scénarios permet au concepteur de spécifier l'ordre dans lequel il *veut* que certains scénarios soient exécutables contrairement à l'intégration implicite où l'ordre d'exécution est déterminé par l'algorithme d'intégration. Les *voeux* du concepteur sont exprimés sous forme de contraintes d'ordonnancement que nous appelons *directives d'intégration explicite*. Par exemple, le concepteur peut spécifier qu'un scénario soit exécuté après tel autre scénario. L'intégration explicite de ces scénarios consiste à déterminer si les scénarios fournis satisfont les contraintes d'ordonnancement spécifiées par le concepteur. Dans le cas où les contraintes d'ordonnancement ne sont pas satisfaites, nous proposons un algorithme qui sur-contraint les scénarios donnés dans le but de satisfaire les contraintes d'ordonnancement. Si les contraintes d'ordonnancement sont satisfaisables, notre algorithme permet d'aboutir à de nouveaux scénarios dérivés des anciens et réalisant les voeux du concepteur. Ensuite, ces nouveaux scénarios seront intégrés en utilisant les modes implicites d'intégration puisque l'ordonnancement est déjà satisfait.

La section 8.1 définit la syntaxe et la sémantique des directives d'intégration explicite. Les directives d'intégration sont exprimées à l'aide d'opérateurs représentant des contraintes d'ordonnancement. La section 8.2 expose le problème de la satisfaction des directives d'intégration explicite. La section 8.3 décrit notre méthode de résolution du problème de la satisfaction des directives d'intégration explicites. La section 8.4 présente un exemple qui illustre l'application de notre méthode de résolution du problème de satisfaction des directives d'intégration explicite.

8.1 Spécification des directives d'intégration explicite

Les directives d'intégration explicite représentent des déclarations qui sont fournies par le concepteur. Chaque directive d'intégration explicite est une contrainte d'ordonnancement à respecter sur l'ordre d'exécution de certains scénarios. Une directive relie deux scénarios en utilisant l'un des opérateurs d'intégration explicite. Nous proposons deux opérateurs d'intégration explicite, l'un est l'opérateur séquentiel, l'autre est l'opérateur alternative. L'opérateur séquentiel contraint deux scénarios de telle sorte que l'exécution de l'un soit possible après l'exécution de l'autre (figure 8.1(a)) alors que l'opérateur alternative s'applique sur deux scénarios pour que le système ait comme alternative, dans une situation, l'exécution de l'un ou l'autre parmi ces deux scénarios (figure 8.1(b)).



(a) Opérateur séquentiel : $sc_1 \rightsquigarrow sc_2$ (b) Opérateur alternative : $sc_1 | sc_2$

FIG. 8.1 – Opérateurs d'intégration explicite de scénarios

Dans la suite de cette section, nous nous donnons les définitions des opérateurs et des directives d'intégration explicite.

8.1.1 Opérateurs d'intégration explicite

Un opérateur d'intégration explicite permet de formuler une contrainte d'ordonnancement sur deux scénarios en un utilisant un opérateur d'ordonnancement. Nous commençons par définir l'opérateur séquentiel et l'opérateur alternative.

Définition 8.1. *Opérateur séquentiel*
Soient sc_1 et sc_2 deux scénarios consistants appartenant à S. Nous notons "$\rightsquigarrow$" l'opérateur séquentiel. La directive $sc_1 \rightsquigarrow sc_2$ est la contrainte d'ordonnancement séquentiel entre les scénarios sc_1 et sc_2. La directive $sc_1 \rightsquigarrow sc_2$ est satisfaite ssi les scénarios sc_1 et sc_2 vérifient les conditions suivantes :

(i) *les scénarios sc_1 et sc_2 sont consistants*

(ii) $[sc_1.\psi_N] \subset [sc_2.\psi_{N_0}]$

(iii) $[sc_1.\varphi_N] \subset [sc_2.\varphi_{N_0}]$

Où $sc_1.\psi_N$ et $sc_1.\varphi_N$ représentent respectivement les deux contraintes de variables et d'horloges définissant le contexte du dernier noeud principal N du scénario sc_1 et où $sc_2.\psi_{N_0}$ et $sc_2.\varphi_{N_0}$ représentent respectivement les contraintes de variables et d'horloges définissant le contexte du premier noeud principale N_0 du scénario sc_2.

Dans la définition précédente, les conditions (ii) et (iii) expriment que l'exécution du scénario sc_2 n'est permise qu'après une exécution complète du scénario sc_1. Nous rappelons qu'une exécution complète d'un scénario correspond à une exécution de toutes ses actions principales commençant par la première jusqu'à la dernière. Par conséquent, l'exécution complète du scénario sc_1 est une condition nécessaire mais non toujours suffisante pour que l'exécution du scénario sc_2 puisse avoir lieu.

Définition 8.2. *Opérateur alternative*
Soient sc_1 et sc_2 deux scénarios consistants appartenant à S. Nous notons "$|$" l'opérateur alternative. La directive $sc_1|sc_2$ est la contrainte d'ordonnancement alternative entre les scénarios sc_1 et sc_2. La directive $sc_1|sc_2$ est satisfaite ssi les scénarios sc_1 et sc_2 vérifient les conditions suivantes :

(i) *les scénarios sc_1 et sc_2 sont consistants*

(ii) $sc_1.\psi_{N_0} = sc_2.\psi_{N_0}$

Où $sc_1.\psi_{N_0}$ et $sc_2.\psi_{N_0}$ représentent respectivement les contraintes de variables du contexte du premier noeud principal N_0 des scénarios sc_1 et sc_2

Dans la définition 8.2, la condition $sc_1.\psi_{N_0} = sc_2.\psi_{N_0}$ exprime que la contrainte $sc_1|sc_2$ exige que les scénarios sc_1 et sc_2 aient les mêmes pré-conditions concernant les variables. Ainsi, lorsque

l'état du système satisfait les pré-conditions des scénarios sq et sc_2, l'exécution du système pourra alors suivre l'un ou l'autre des deux scénarios selon la stimulation de l'environnement du système.

Remarque 8.1. *La directive $sc|sc$ est toujours satisfaite si le scénario sc est consistant.*

8.1.2 Déclaration des directives d'intégration explicite

Le concepteur déclare des contraintes sur l'ordonnancement de certains scénarios en fournissant un ensemble de directives Dir où chaque directive est de la forme $sq\#sc_2$ telle que $\# \in \{\rightsquigarrow, |\}$ et sc_1, sc_2 sont des scénarios de S. En général, les scénarios de S, tels qu'ils sont fournis par le concepteur, ne satisfont pas les directives d'intégration explicite de Dir selon les conditions des définitions 8.1 et 8.2. Pour satisfaire les exigences du concepteur qui sont spécifiées dans Dir, nous allons sur-contraindre les scénarios qui interviennent dans une directive de Dir pour aboutir à des scénarios dérivés des premiers et qui satisfont toutes les directives de Dir. Les exemples suivants illustrent cette opération.

Exemple 8.1. *Pour cette exemple nous définissons les ensembles V, H, Ev, S et Dir comme suit :*

- $V = \{v_1, v_2\}$ *tel que* $dom(v_1) = \{OFF, ON\}$ *et* $dom(v_2) = \{HIGH, LOW\}$
- $H = \{h\}$
- $Ev = \{evt_1, evt_2\}$
- $S = \{sc_1, sc_2, sc_3\}$
- $Dir = \{sc_1 \rightsquigarrow sc_2\}$

Nous supposons que les scénarios sc_1 et sc_2 sont de simples scénarios qui ne comportent qu'une seule action. Les scénarios sc_1 et sc_2 sont décrits par :

scénario sc_1	*scénario* sc_2
$\psi_{sc_1.a_{00}} = (v_1 = OFF)$	$\psi_{sc_2.a_{00}} = (v_2 = LOW)$
$\varphi_{sc_1.a_{00}} = (h \leq 3)$	$\varphi_{sc_2.a_{00}} = True$
$l_{sc_1.a_{00}} = evt_1$	$l_{sc_2.a_{00}} = evt_2$
$\phi_{sc_1.a_{00}} = (h \leq 3)$	$\phi_{sc_2.a_{00}} = True$
$\delta_{sc_1.a_{00}} = (v_1 := ON)$	$\delta_{sc_2.a_{00}} = (v_2 := HIGH)$
$\lambda_{sc_1.a_{00}} = \{h\}$	$\lambda_{sc_2.a_{00}} = \{\}$

Dans cet exemple, pour satisfaire $Dir = \{sc_1 \rightsquigarrow sc_2\}$*, qui exprime que l'exécution de* sc_2 *n'est possible qu'après l'exécution de* sc_1*, il faut sur-contraindre le scénario* sc_2 *avec la contrainte* $(v_1 = ON)$ *qui résulte de l'exécution de* sc_1*. En sur-contraignant* $sc2$ *avec la contrainte* $(v_1 = ON)$*, nous aboutissons à un scénario* sc'_2 *tel que la directive* $sc_1 \rightsquigarrow sc'_2$ *est satisfaite. Le scénario* sc'_2 *est défini par :*

scénario sc'_2
$\psi_{sc'_2.a_{00}} = \psi_{sc_2.a_{00}} \wedge (v_1 = ON) = (v_2 = LOW) \wedge (v_1 = ON)$
$\varphi_{sc'_2.a_{00}} = \varphi_{sc_2.a_{00}} = True$
$l_{sc'_2.a_{00}} = l_{sc2.a_{00}} = evt_2$
$\phi_{sc'_2.a_{00}} = \phi_{sc_2.a_{00}} = True$
$\delta_{sc'_2.a_{00}} = \delta_{sc_2.a_{00}} = (v_2 := HIGH)$
$\lambda_{sc'_2.a_{00}} = \lambda_{sc_2.a_{00}} = \{\}$

Exemple 8.2. *Nous considérons pour cette exemple la même description du système que l'exemple 8.1 sauf pour Dir pour lequel nous supposons que* $Dir = \{sc_1 | sc_2\}$*.*

La directive $sc_1|sc_2$ *n'est pas satisfaite. En sur-contraignant les scénarios* sc_1 *avec la contrainte* $(v_2 = LOW)$ *et le scénario* sc_2 *avec la contrainte* $(v_1 = OFF)$*, nous obtenons deux scénarios* sc'_1 *et* sc'_2 *(décrits ci-après) tels que la directive* $sc'_1|sc'_2$ *est satisfaite.*

scénario sc'_1	*scénario* sc'_2
$\psi_{sc'_1.a_{00}} = \psi_{sc_1.a_{00}} \wedge (v_2 = LOW)$	$\psi_{sc'_2.a_{00}} = \psi_{sc_2.a_{00}} \wedge (v_1 = OFF)$
$\varphi_{sc'_1.a_{00}} = \varphi_{sc_1.a_{00}} = (h \leq 3)$	$\varphi_{sc'_2.a_{00}} = \varphi_{sc_2.a_{00}} = True$
$l_{sc'_1.a_{00}} = l_{sc_1.a_{00}} = evt_1$	$l_{sc'_2.a_{00}} = l_{sc_2.a_{00}} = evt_2$
$\phi_{sc'_1.a_{00}} = \phi_{sc_1.a_{00}} = (h \leq 3)$	$\phi_{sc'_2.a_{00}} = \phi_{sc_2.a_{00}} = True$
$\delta_{sc'_1.a_{00}} = \delta_{sc_1.a_{00}} = (v_1 := ON)$	$\delta_{sc'_2.a_{00}} = \delta_{sc_2.a_{00}} = (v_2 := HIGH)$
$\lambda_{sc'_1.a_{00}} = \lambda_{sc_1.a_{00}} = \{h\}$	$\lambda_{sc'_2.a_{00}} = \lambda_{sc_2.a_{00}} = \{\}$

Les pré-conditions sur les variables des scénarios sc'_1 *et* sc'_2 *sont les mêmes. Ils correspondent à la conjonction des pré-conditions sur les variables des scénario* sc_1 *et* sc_2*.*

Les exemples précédents permettent d'avoir une idée de comment satisfaire les directives d'intégration explicite. Le problème de la satisfaction des directives d'intégration explicites consiste

à déterminer une contrainte de variables (si elle existe) pour chaque scénarios afin que les scénarios sur-contraints avec ces contraintes satisfont les directives spécifiées.

Pour formaliser l'opération de sur-contraindre un scénario, nous définissons une fonction $scen$ qui prend comme paramètres un scénario $sc \in S$ et une contrainte de variables $\psi \in \Psi(V)$. Nous notons $scen(sc, \psi)$ le scénario dérivé en sur-contraignant sc avec la contrainte de variables ψ.

Définition 8.3. *Soient sc un scénario et ψ une contrainte de variables. La fonction scen associe au scénario sc et à la contrainte ψ un scénario $sc' = scen(sc, \psi)$. Le scénario sc' a la même structure que sc en terme du nombre de noeuds et du nombre d'actions par noeud. Ainsi, chaque action de sc admet un homologue dans $sc' = scen(sc, \psi)$. Chaque action du scénario sc' est identique à son homologue sauf pour la premier action principale que nous définissons comme suit. Soit $sc'.a_{00}$ la première action principale[1] du scénario sc'. L'action $sc'.a_{00}$ est définie en fonction $sc.a_{00}$ par :*

- $\psi_{sc'.a_{00}} \stackrel{def}{=} \psi_{sc.a_{00}} \wedge \psi$
- $(\varphi_{sc'.a_{00}}, \varphi'_{sc'.a_{00}}, l_{sc'.a_{00}}, \lambda_{sc'.a_{00}}) \stackrel{def}{=} (\varphi_{sc.a_{00}}, \varphi'_{sc.a_{00}}, l_{sc.a_{00}}, \lambda_{sc.a_{00}})$

Dans l'exemple 8.1, la contrainte $(v_1 = ON)$ caractérise l'état du système après l'exécution du scénario sc_1. Pour satisfaire la directive $sc_1 \rightsquigarrow sc_2$, nous avons sur-contraint le scénario sc_2 avec la contrainte $(v_1 = ON)$, ce qui a donné lieu au scénario sc'_2. Le scénario sc'_2 est exactement $scen(sc_2, (v_1 = ON))$ et la directive $sc_1 \rightsquigarrow sc'_2$ est satisfaite.

Par conséquent en généralisant ce raisonnement, nous pouvons conclure que pour satisfaire un ensemble de directive $Dir = \{sc_1 \rightsquigarrow sc_2\}$, il suffit de considérer la contrainte de variables qui caractérise l'état du système après l'exécution du scénario sc_1. Cette contrainte est en fait la contrainte de variables du contexte du dernier noeud principal du scénario sc_1. En supposant que N est le dernier principal du scénario sc_1, cette contrainte est $\psi_{sc_1.N}$. Les scénarios sc_1 et $sc'_2 = scen(sc_2, (v_1 = ON))$ satisfont la directive $sc_1 \rightsquigarrow sc'_2$ si sc_1 et sc'_2 sont consistants.

Dans l'exemple 8.2, la satisfaction de $Dir = \{sc_1 | sc_2\}$ a nécessité de sur-contraindre sc_1 avec $(v_2 = LOW)$ et sc_2 avec $(v_1 = OFF)$. Les scénarios sc'_1 et sc'_2 sont respectivement exactement $scen(sc_1, (v_2 = LOW))$ et $scen(sc_2, (v_1 = OFF)$.

[1] cf. les définitions 6.1 et 6.2

Le raisonnement déduit de l'exemple 8.2 est généralisable pour la satisfaction d'un ensemble de directives de la forme $Dir = \{sc_1|sc_2\}$. La satisfaction de Dir dans ce cas consiste à surcontraindre les scénarios sc_1 et sc_2 respectivement avec les contraintes $\psi_{sc_2.N_0}$ et $\psi_{sc_1.N_0}$ qui sont respectivement les contraintes de variables du contexte des premiers noeuds principaux des scénarios sc_1 et sc_2 respectivement. L'ensemble de directives Dir est satisfait si les scénarios $sc'_1 = scen(sc_1, \psi_{sc_2.N_0})$ et $sc'_2 = scen(sc_2, \psi_{sc_2.N_0})$ sont consistants. Nous remarquons que sc'_1 et sc'_2 ont la même pré-condition sur les variables et qui est $\psi_{sc_2.N_0} \wedge \psi_{sc_1.N_0}$.

Le cas général du problème de la satisfaction d'un ensemble de directives d'intégration explicite est plus complexe. Dans la sous-section suivante, nous définissons ce problème et proposons une solution.

8.2 Satisfaction des directives d'intégration explicite

Dans la section précédente nous avons vu, à travers quelques exemples, comment satisfaire une directive d'intégration explicite. Dans le cas général, satisfaire un ensemble de directives d'intégration explicite est un problème plus complexe que les exemples que nous avons présentés. En effet, l'ensemble de directives est considéré comme un système de contraintes sur l'ordonnancement des scénarios qu'il faut satisfaire *simultanément.*

Définition 8.4. PSED *Le Problème de la Satisfaction d'un Ensemble de Directive (PSED) Dir consiste à déterminer pour chaque scénario sc, qui intervient dans une directive de Dir, une contrainte de variable ψ_{sc} (si elle existe) telle que* pour toute directive $sc_1 \# sc_2$ *appartenant à Dir, la directive $scen(sc_1, \psi_{sc_2}) \# scen(sc_2, \psi_{sc_2})$ est satisfaite.*

Afin de faire ressortir les points à traiter pour résoudre le problème PSED, nous considérons quelques cas particuliers.

8.2.1 Opérateur de séquencement

Nous remarquons que pour satisfaire l'ensemble $Dir = \{sc_1 \rightsquigarrow sc_2, sc_2 \rightsquigarrow sc_3\}$, il faut déterminer une contrainte ψ_{sc2} pour que les scénarios sc_1 et $sc'_2 = scen(sc_2, \psi_{sc_2})$ satisfassent la directive $sc_1 \rightsquigarrow sc_2$. Pour déterminer la contrainte ψ_{sc_3} qui devrait contraindre sc_3, il faut

considérer la directive $sc_2' \rightsquigarrow sc_3$. De cet exemple nous remarquons que les contraintes ψ_{sc} se propagent à travers les scénarios. Par conséquent, nous construisons un graphe dirigé que nous appelons *graphe d'intégration explicite* (GIE) GIE et qui permet la propagation des contraintes ψ_{sc} à travers les scénarios. Les arcs dirigés du GIE sont étiquetés par les scénarios. Chaque arc relie entre deux noeuds dont le premier représente le début du scénario et le deuxième sa fin. La figure 8.2 montre le GIE associé à $Dir = \{sc_1 \rightsquigarrow sc_2, sc_2 \rightsquigarrow sc_3\}$ dans cet exemple. Le problème PSED consiste à affecter à chaque noeud x, y et z du GIE de la figure 8.2 une contrainte de variables ψ_x, ψ_y et ψ_z respectivement telle que les scénarios $scen(sc_1, \psi_x)$, $scen(sc_2, \psi_y)$ et $scen(sc_3, \psi_z)$ représentent la solution, si elle existe, du PSED de Dir.

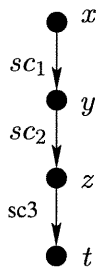


FIG. 8.2 – GIE pour $Dir = \{sc_1 \rightsquigarrow sc_2, sc_2 \rightsquigarrow sc_3\}$

8.2.2 Opérateur d'alternative

Le GIE que nous associons à l'ensemble de directive $Dir = \{sc_1|sc_2, sc_2|sc_3\}$ est représenté à la figure 8.3. Le problème PSED , dans ce cas, consiste à affecter à au noeud x du GIE de la figure 8.2 une contrainte de variables ψ_x telle que les scénarios $scen(sc_1, \psi_x)$, $scen(sc_2, \psi_x)$ et $scen(sc_3, \psi_x)$ représentent la solution, si elle existe, du PSED de Dir.

8.2.3 Graphe d'intégration explicite avec cycle

La construction du GIE est utile surtout lorsque celui-ci comporte des cycles. C'est le cas de l'ensemble de directives $Dir = \{sc_1 \rightsquigarrow sc_2, sc_2 \rightsquigarrow sc_3, sc_3 \rightsquigarrow sc_1\}$ dont le GIE est représenté par la figure 8.4. Les contraintes ψ_x, ψ_y et ψ_z sont initialisées avec les contraintes

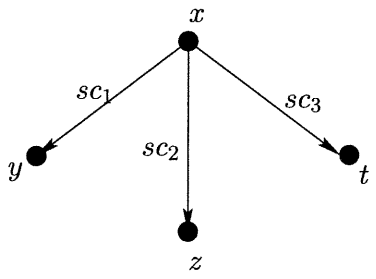


FIG. 8.3 – GIE pour $Dir = \{sc_1|sc_2, sc_2|sc_3\}$

$\psi_{sc_1.N_0}$, $\psi_{sc_2.N_0}$ et $\psi_{sc_3.N_0}$ respectivement. Les contraintes ψ_x, ψ_y et ψ_z sont propagées à travers le cycle (sc_1, sc_2, sc_3, sc_1) du GIE de la figure 8.4 en utilisant les affectations suivantes :

$$\psi_x := \psi_x \wedge \psi_{scen(sc_3,\psi_z).N} \tag{8.1}$$

$$\psi_y := \psi_y \wedge \psi_{scen(sc_1,\psi_x).N} \tag{8.2}$$

$$\psi_z := \psi_z \wedge \psi_{scen(sc_2,\psi_y).N} \tag{8.3}$$

Dans les instructions (8.1), (8.2) et (8.3), nous désignons pas $\psi_{scen(sc_i,\psi).N}$ la contrainte de variables du contexte du dernier noeud principal N du scénario $scen(sc_i, \psi)$. Les instructions (8.1), (8.2) et (8.3) répéter autant de fois que nécessaire jusqu'à ce ψ_x, ψ_y et ψ_z deviennent stables. À ce moment, la solution du PSED de Dir, si elle existe, est constituée des scénarios $scen(sc_1, \psi_x)$, $scen(sc_2, \psi_y)$ et $scen(sc_3, \psi_z)$.

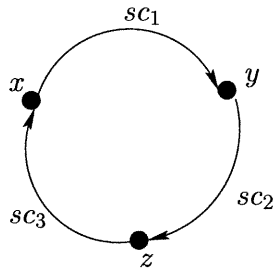


FIG. 8.4 – GIE pour $Dir = \{sc_1 \rightsquigarrow sc_2, sc_2 \rightsquigarrow sc_3, sc_3 \rightsquigarrow sc_1\}$

8.2.4 Cas particuliers de graphes d'intégration explicite

Les ensembles de directives $Dir_1 = \{sc_1 \rightsquigarrow sc_2, sc_2|sc_3\}$ et $Dir_2 = \{sc_1 \rightsquigarrow sc_2, sc_1 \rightsquigarrow sc_3\}$ représentent deux cas particuliers intéressants à considérer avant d'énoncer la définition générale du GIE d'un ensemble de directives. Sur le plan de la sémantique, Din_1 et Dir_2 sont différents et donc ils doivent avoir des GIEs qui reflètent leurs sémantiques respectives. D'après les définitions 8.2 et 8.1, le GIE de Dir_1 est celui représenté à la figure 8.5(a). L'ensemble de directives Dir_1 spécifie que les scénarios sc_2 et sc_3 ont les mêmes pré-conditions sur les variables et que l'exécution de sc_1 est nécessaire pour exécuter sc_2 et donc pour sc_3 aussi. Cette sémantique est bien représentée par le GIE de la figure 8.5(a).

Les directives spécifiées dans Dir_2 expriment que l'exécution du scénario sc_1 est aussi bien nécessaire pour l'exécution du scénario sc_2 que pour le scénario sc_3. Cependant, nous n'avons pas forcément que les scénarios sc_2 et sc_3 doivent avoir les mêmes pré-conditions. Par conséquent le graphe de la figure 8.5(a) ne peut représenter le GIE de Dir_2. Pour éviter cette confusion, nous proposons un artifice de calcul qui consiste à dupliquer le scénario sq en deux copies $sc_1^{(0)}$ et $sc_1^{(1)}$. Nous transformons l'ensemble de directive Dir_2, qui porte à la confusion, en un ensemble $Dir'_2 = \{sc_1^{(0)} \rightsquigarrow sc_2, sc_1^{(1)} \rightsquigarrow sc_3, sc_1^{(0)}|sc_1^{(1)}\}$. Les copies du scénarios sc_1 permettent de maintenir les directives de la forme $(sc_1 \rightsquigarrow .)$. La directive de la forme $sc_1^{(0)}|sc_1^{(1)}$, que nous avons ajoutée dans Dir'_2, sert à exprimer que les scénarios $sc_1^{(0)}$ et $sc_1^{(1)}$ doivent avoir les mêmes pré-conditions puisqu'ils sont des copies du même scénario sq. Dir'_2 est appelée forme canonique de Dir_2. Le GIE de Dir_2 est construit à partir de Dir'_2 et représenté à la figure 8.5(b).

8.3 Résolution du PSED

Dans cette section nous commençons par définir certains éléments qui vont entrer dans la résolution du PSED. Ensuite, nous regroupons ces éléments pour énoncer la méthode de résolution du PSED.

8.3.1 Graphe d'intégration explicite

La forme canonique d'un ensemble de directives d'intégration explicite capture sa sémantique. Ainsi la construction du GIE d'un ensemble de directives se base sur sa forme canonique. Tous les ensembles de directives qui ont la même forme canonique ont le même GIE. Nous notons Dir' la

(a) GIE de Dir_1 (b) GIE de Dir_2

FIG. 8.5 – Opérateurs d'intégration explicite de scénarios

forme canonique d'un ensemble de directives Dir.

Définition 8.5. *Forme canonique d'un ensemble de directives*
Soit Dir un ensemble de directives et Dir' sa forme canonique. Pour chaque scénario sc, soit $k_{sc} > 1$ (s'il existe) le plus grand entier vérifiant :

$$\exists sc_1, sc_2, \ldots sc_{k_{sc}} \in S \,.\, \forall 1 \leq i \leq k_{sc}\ (sc \rightsquigarrow sc_i) \in Dir) \tag{8.4}$$

le scénario sc est dupliqué $k_{sc} - 1$ fois. Nous supposons que $sc^{(1)}, sc^{(2)}, \ldots, sc^{(k-1)}$ sont les duplicatas du scénario sc et que $sc^{(0)}$ représente sc.
Dir' est obtenu en remplaçant la directive $sc \rightsquigarrow sc_i \in Dir$ par les deux directives $(sc^{(i-1)} \rightsquigarrow sc_i)$ et $(sc^{(i-1)} \mid sc)$ pour chaque sc qui vérifie l'équation (8.4) *et chaque $1 \leq i \leq k_{sc}$.*

Dans la définition 8.5, les directives de Dir qui ne vérifient pas la condition (8.4) restent inchangées dans Dir'.

Exemple 8.3.

- *Pour $Dir_1 = \{sc_1 \rightsquigarrow sc_2, sc_3 | sc_4\}$, nous obtenons $Dir'_1 = Dir_1$.*
- *Pour $Dir_2 = \{sc_1 \rightsquigarrow sc_2, sc_1 \rightsquigarrow sc_3, sc_1 \rightsquigarrow sc_5, sc_3 | sc_4\}$, nous obtenons $Dir'_2 = \{sc_1^{(0)} \rightsquigarrow sc_2, sc_1^{(0)} | sc_1, sc_1^{(1)} \rightsquigarrow sc_3, sc_1^{(1)} | sc_1, sc_1^{(2)} \rightsquigarrow sc_5, sc_1^{(2)} | sc_1, sc_3 | sc_4\}$. Nous rappelons $sc_1^{(0)} = sc_1$ par convention. L'expression de Dir'_2 peut être simplifiée.*
- *Pour $Dir_3 = \{sc_1 \rightsquigarrow sc_2, sc_1 \rightsquigarrow sc_3, sc_2 \rightsquigarrow sc_5, sc_2 \rightsquigarrow sc_4, sc_3 | sc_4, sc_4 \rightsquigarrow sc_1\}$, Nous obtenons, après simplification, $Dir'_3 = \{sc_1 \rightsquigarrow sc_2, sc_1^{(1)} \rightsquigarrow sc_3, sc_1^{(1)} | sc_1, sc_2 \rightsquigarrow$*

$sc_5, sc_2^{(1)} \rightsquigarrow sc_4, sc_2^{(1)} | sc_2, sc_3 | sc_4, sc_4 \rightsquigarrow sc_1\}$.

La duplication des scénarios est juste un artifice qui permet de construire le GIE d'un ensemble de directives d'intégration explicite dans le but de résoudre le PSED qui lui est associé. Nous avons représenté dans les figure 8.2, 8.3, 8.4 et 8.5, les GIEs respectifs associés à certains ensembles de directives particuliers. Nous énonçons maintenant la définition générale du GIE associé à un ensemble de directives.

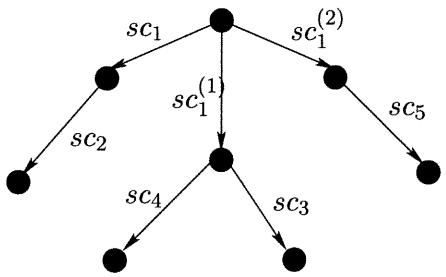
Définition 8.6. *Graphe d'intégration explicite (GIE)*
Soit Dir un ensemble de directives. Le GIE associé à Dir est le graphe dirigé que nous notons $(\mathbb{V}_{Dir}, \mathbb{E}_{Dir})$ où $\mathbb{V}_{Dir}$ et $\mathbb{E}_{Dir}$ sont respectivement son ensemble des noeuds et son ensemble d'arcs.

Chaque scénario qui intervient dans une directive de Dir' étiquette un et seul arc appartenant à $\mathbb{E}_{Dir}$. Un arc étiqueté par un scénario sc est noté par $(x, sc, y) \in \mathbb{E}_{Dir}$ où ses bouts x et y sont des noeuds appartenant à $\mathbb{V}_{Dir}$. Les noeuds x et y modélisent respectivement le début et la fin du scénario sc. Les arcs de $\mathbb{E}_{Dir}$ sont connectés en partageant des bouts communs dictés par les directives de Dir' :

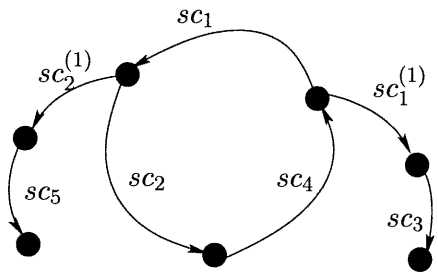
- *Pour chaque directive $(sc_1 \rightsquigarrow sc_2) \in Dir'$ le noeud d'arrivée de l'arc étiqueté par sc_1 et le noeud de départ de l'arc étiqueté par sc_2 sont le même. Nous écrivons $(x, sc_1, y) \in \mathbb{E}_{Dir}$ et $(y, sc_2, z) \in \mathbb{E}_{Dir}$ tels que x,y et z sont des noeuds de $\mathbb{V}_{Dir}$.*
- *Pour chaque directive $(sc_1 | sc_2) \in Dir'$, les arcs étiquetés par sc_1 et sc_2 partagent le même noeud de départ. Nous pouvons écrire $(x, sc_1, y) \in \mathbb{E}_{Dir}$ et $(x, sc_2, z) \in \mathbb{E}_{Dir}$ tels que x,y et z sont des noeuds de $\mathbb{V}_{Dir}$.*

Nous remarquons que la définition 8.6 se base sur la forme canonique Dir' d'un ensemble de directives Dir pour définir son GIE. L'artifice de duplication des scénarios a permis d'associer à chaque scénarios de Dir' un arc unique dans le GIE. Les arcs correspondants aux deux scénarios d'une directive de Dir' partagent l'un de leurs deux bouts respectives.

La figure 8.6 montre les GIEs associés respectivement à Dir_2 et Dir_3 qui sont définis dans l'exemple 8.3. La définition 8.6 généralise le traitement que nous avons exposé pour aboutir aux GIE des figures 8.2, 8.3, 8.4 et 8.5.



(a) GIE de Dir_2 de l'exemple 8.3



(b) GIE de Dir_3 de l'exemple 8.3

FIG. 8.6 – Exemples de GIEs

8.3.2 Calcul des contraintes associées aux noeuds d'un graphe d'intégration explicite

Nous rappelons que nous avons défini le GIEd'un ensemble de directive dans le but d'associer à chaque noeud du graphe une contrainte de variables. La solution (si elle existe) du PSED (définition 8.4) est obtenue en contraignant chaque scénario avec la contrainte de variable qui est associée au noeud de départ de l'arc de ce scénario dans le GIE. La résolution du PSED consiste à déterminer pour chaque noeud x du GIE une contrainte de variables ψ_x pour que les scénarios de la forme $scen(sc, \psi_x)$, telle que (x, sc, y) est un arc du GIE, représentent la solution, si elle existe, du PSED. Ainsi, la contrainte ψ_{sc} décrite dans la définition 8.4 sera exactement la contrainte ψ_x telle (x, sc, y) est un arc du GIE.

Chaque arc (x, sc, y) du GIE permet la propagation de la contrainte ψ_x, à travers le scénario sc vers le noeud y. Cette propagation est assurée par la fonction $propagate$ qui prend en paramètre une contrainte de variables ψ et un scénario sc. Ainsi, en posant $sc' = scen(sc, \psi)$, nous définissons la fonction $propagate$ par :

$$propagate(\psi, sc) \stackrel{def}{=} \psi_{sc'.N} \tag{8.5}$$

où $\psi_{sc'.N}$ est la contrainte de variables du contexte du dernier noeud principal N du scénario $sc' = scen(sc, \psi)$.

Pour déterminer la contrainte de chaque noeud x du GIE, nous initialisons ψ_x par la conjonction des pré-conditions des scénarios qui ont le noeud x comme noeud de départ dans le GIE. Après, la phase d'initialisation, nous commençons la phase de propagation. Pour chaque arc (x, sc, y) du GIE, nous propageons la contrainte ψ_x à travers le scénario sc pour raffiner la contrainte courante ψ_y par $\psi_y \wedge propagate(\psi, sc)$. En suite, si ψ_y est modifiée, elle va être propagée à son tour à travers tous les scénarios qui partent du noeud y. Ainsi de suite, cette phase est répétée jusqu'à la stabilisation de toutes les contraintes des noeuds du GIE. Pour prouver la terminaison de cet algorithme, nous proposons pour chaque noeud du GIE une suite de contraintes dont les termes convergent vers la contrainte du noeud.

Étant donné un ensemble de directives d'intégration explicite Dir, nous représentons la contrainte ψ_x d'un noeud x du GIE de Dir comme étant la limite de la suite convergeante $(\psi_x)_n$ définie par :

$$\begin{cases} (\psi_x)_0 = \bigwedge\limits_{(x,sc,y)\in\mathbb{E}_{Dir}} \psi_{sc.N_0} \\ (\psi_x)_{i+1} = (\psi_x)_i \ \wedge \bigwedge\limits_{(y,sc,x)\in\mathbb{E}_{Dir}} propagate((\psi_y)_i, sc) \end{cases}$$

La suite $(\psi_x)_n$ est décroissante car après chaque itération le terme de la suite soit il reste inchangé, soit il est raffiné. Pour tout nombre entier i, nous remarquons que $[(\psi_x)_i] \subseteq [(\psi_x)_{i+1}]$. La suite $(\psi_x)_n$ est bornée car les domaines des variables du système sont discrets et finis. Nous savons que toute suite décroissante bornée converge donc la suite $(\psi_x)_n$ est convergente. De plus, les valeurs possibles des termes de la suite $(\psi_x)_n$ sont finis. Nous concluons que la suite $(\psi_x)_n$ est stationnaire après un certain rang. Donc l'algorithme de calcul des contrainte des noeuds du GIE se termine. Nous décrivons à la figure 8.7 une version itérative de l'algorithme de calcul des contraintes des noeuds qui est implémenté par la procédure $Calcul_Contraintes_$GIE(Dir).

8.3.3 Méthode de résolution du problème de la satisfaction d'un ensemble de directives

Nous regroupons les résultats des sections 8.3.2 et 8.3.1 pour proposer notre méthode de résolution du PSED. Les phases du processus de résolution du PSEDsont représentées à la figure 8.8. Les données du PSEDsont un ensemble de directives d'intégration explicite Dir et la

procédure $Calcul\_Contraintes\_\text{GIE}(Dir)$

pour chaque vertex $x \in \mathbb{V}_{Dir}$

$\varphi_x := \bigwedge_{(x,sc,\_)\in\mathbb{E}_{Dir}} \varphi_{sc.N1}$

$\psi_x := \bigwedge_{(x,sc,\_)\in\mathbb{E}_{Dir}} \psi_{sc.N1}$

let $Z := \{x \in \mathbb{E}_{Dir} \mid \exists y, sc \,.\, (x, sc, y) \in \mathbb{E}_{Dir}\}$

tant que $Z \neq \emptyset$ **faire**

choisir $x \in Z$

$Z := Z - \{x\}$

pour chaque $(x, sc, y) \in \mathbb{E}_{Dir}$

$(\psi, \varphi) := propagate(\psi_x, \varphi_x, sc)$

si $(\psi_y, \varphi_y) \neq (\psi, \varphi)$ **alors**

$\psi_y := \psi_y \wedge \psi$

$\varphi_y := \varphi_y \wedge \varphi$

$Z := Z \cup \{y\}$

FIG. 8.7 – Algorithme de calcul des contraintes des noeuds d'un GIE

description des scénarios contraints par les directives de Dir. La phase 1 permet le calcul de la forme canonique Dir' de Dir. Le résultat de la phase 2 est la construction du GIE de Dir en se basant sur sa forme canonique Dir'. La phase 3 permet de calculer les contraintes des noeuds du GIE de Dir en utilisant la procédure $Calcul_Contraintes_$GIE(Dir).

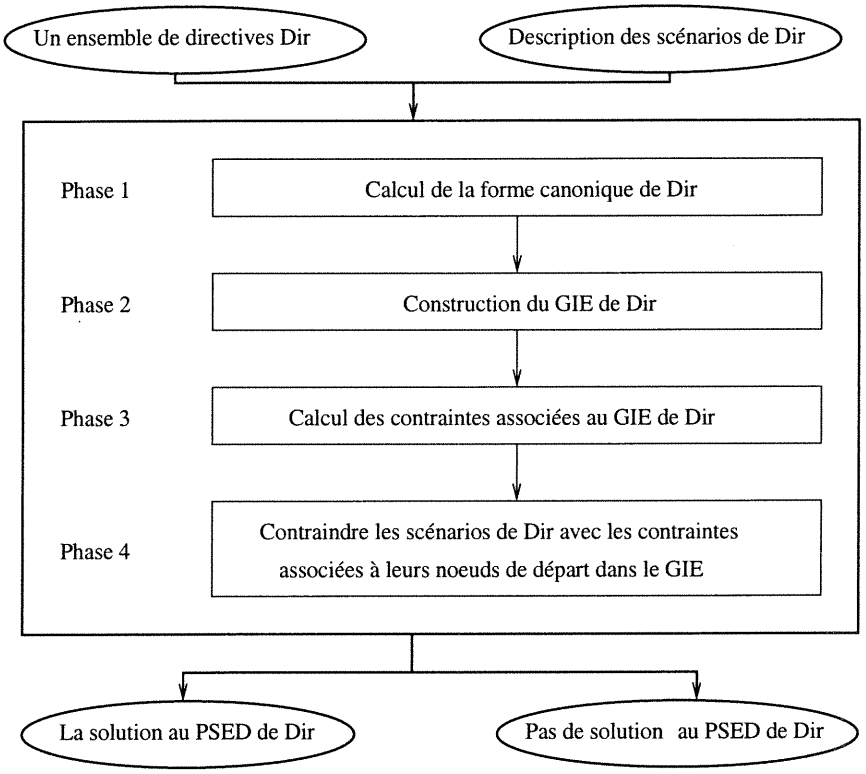


FIG. 8.8 – Le processus de la résolution du PSED

S'il existe un noeud $x \in \mathbb{V}_{Dir}$ tel que $\psi_x = False$, le PSED de Dir n'a pas de solution. Par contre, si l'ensemble de directives Dir admet une solution alors elle est obtenu en contraignant les scénarios de Dir avec les contrainte des noeuds du GIE. Pour chaque scénario sc de Dir dont l'arc dans le GIE est $(x, sc, y) \in \mathbb{E}_{Dir}$, nous notons $sc' = scen(sc, \psi_x)$. Les scénarios sc' tels que sc est un scénario de Dir, satisfont les points (ii) des deux définitions 8.1 et 8.2. En effet, pour tout $(sc_1 \rightsquigarrow sc_2) \in Dir$, les scénarios sc'_1 et sc'_2 satisfont le point (ii) de la définition 8.1. De même, pour tout $(sc_1|sc_2) \in Dir$ les scénarios sc'_1 et sc'_2 satisfont le point (ii) de la définition 8.2.

Pour que l'ensemble composé des scénarios sc' constitue une solution du PSED de Dir, il faut vérifier que les conditions suivantes sont satisfaites.

1. Les scénarios sc' doivent être consistants d'après les points (i) des deux définitions 8.1 et 8.2.
2. Pour les scénarios sc_1 et sc_2 tels que $(sc_1 \rightsquigarrow sc_2) \in Dir$, il faut que sc'_1 et sc'_2 vérifient la condition (iii) de la définition 8.1.

Remarque 8.2. *Le point 2., ci-dessus peut être vérifié au départ. Pour $(sc_1 \rightsquigarrow sc_2) \in Dir$, si les scénarios $sc1$ et sc_2 vérifient la condition (iii) de la définition 8.1 alors sc'_1 et sc'_2 la vérifient aussi.*

Remarque 8.3. *Puisque le PSED n'admet pas de solution si l'un des ψ_x a la valeur $False$, la procédure $Calcul\_Contraintes\_GIE$ peut être modifiée pour s'arrêter et signaler qu'il y a pas de solution dès que l'une des valeurs intermédiaires d'un ψ_x est $False$.*

Lorsque les scénarios sc' vérifient les points 1. et 2. précedents, ils représentent la solution du PSED de Dir. L'automate temporisé sécuritaire modélisant le système est obtenu en utilisant un mode d'intégration implicite sur les scénarios sc' avec les scénarios qui ne sont contraints par aucune directive d'intégration explicite.

8.4 Exemple d'application

Nous reprenons l'exemple du système de reconnaissance des clics de souris dans le but d'illustrer l'intégration explicite de scénarios. Nous adoptons un domaine d'application légèrement modifié qui est décrit par la figure 8.9. Nous avons seulement changé de domaine des valeurs de la variable *click*.

Nous allons concevoir trois scénarios sc_1, sc_2 et sc_3 que nous allons contraindre par un ensemble de directives Dir. Les comportements décrits par les scénarios sc_1, sc_2 et sc_3 sont respectivement décrits par les MSCs (a), (b) et (c) de la figure 8.10. Nous remarquons que les préconditions des MSCs MSC (a), (b) et (c) de la figure 8.10 sont moins contraingnantes que celles qui sont indiquées dans les MSC qui leurs correpondent respectivement dans la figure 7.4. Pour le MSC (a) de la figure 8.10, la précondition est $(sel = OFF)$ alors que dans le MSC (a) de la figure 7.4 la

$$Ev = \{SC, DC, DS, FS, Ha, B\}$$
$$V = \{pos, click, sel\}$$
$$dom(pos) = \{HAUT, BAS\}$$
$$dom(click) = \{0, 1, 2\}$$
$$dom(sel) = \{ON, OFF\}$$
$$H = \{h\}$$

FIG. 8.9 – Description du domaine d'application pour le système de reconnaissance des clics de la souris

précondition est $(sel = OFF) \wedge (click = N) \wedge (pos = HAUT)$. Par contre les MSC (b) et (c) de la figure 8.10 n'ont pas de précondition à la difference des MSC (b) et (c) de la figure 7.4.

Les scénarios sc_1, sc_2 et sc_3 sont décrits à la table 8.1. Le scénario sc_4 de la table 8.1 correspond au MSC (d) de la figure 8.10 et traite le cas de la *sélection*. Les scénarios sc_1 , sc_2 et sc_3 sont contraints par $Dir == \{sc_1 \rightsquigarrow sc_2, sc_2|sc_3, sc_2 \rightsquigarrow sc_1, sc_3 \rightsquigarrow sc_3\}$. Le GIE de Dir est donné à figure 8.11. La résolution de PSED de Dir permet de déterminer les contrainte ψ_x et ψ_y :

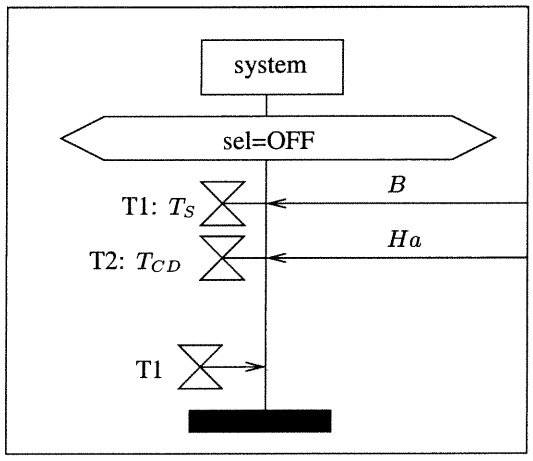
$$\psi_x = (sel = OFF) \wedge (click = 0) \wedge (pos = HAUT)$$

$$\psi_y = (sel = OFF) \wedge (click = 1) \wedge (pos = HAUT)$$

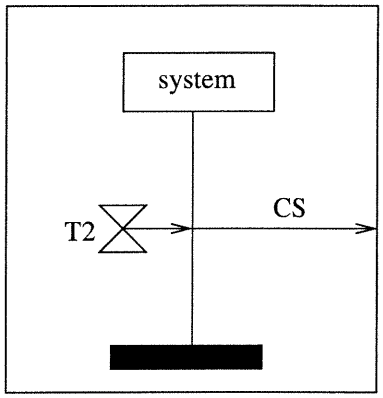
Les nouveaux scénarios $sc'_1 = scen(\psi_x, sc_1)$, $sc'_2 = scen(\psi_y, sc_2)$ et $sc'_3 = scen(\psi_y, sc_3)$ satisfont Dir. L'automate temporisé sécuritaire résultant de l'intégration implicite libre des scénarios sc'_1, sc'_2, sc'_3 et sc_4 est représenté à figure 8.12.

8.5 Conclusion

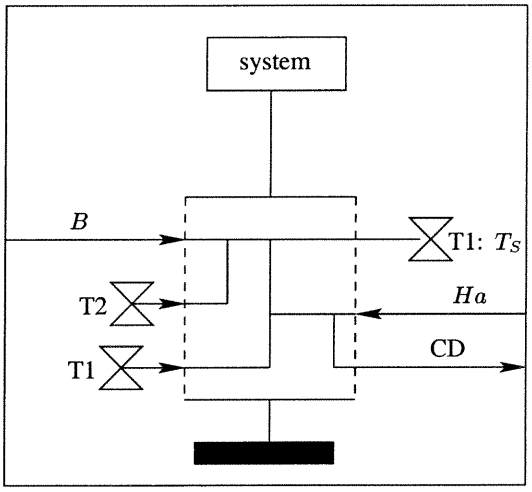
Dans ce chapitre nous avons défini la spécification des directives d'intégration explicite aussi bien au niveau syntaxique que sémantique. Chaque directive d'intégration représente une contrainte d'ordonnancement sur deux scénarios basée sur l'un des opérateurs d'intégration explicite. Nous avons défini deux opérateurs d'intégration explicite : l'opérateur séquentiel et l'opérateur alternative. Ensuite, nous avons discuté de la résolution du PSED pour des ensembles de directives



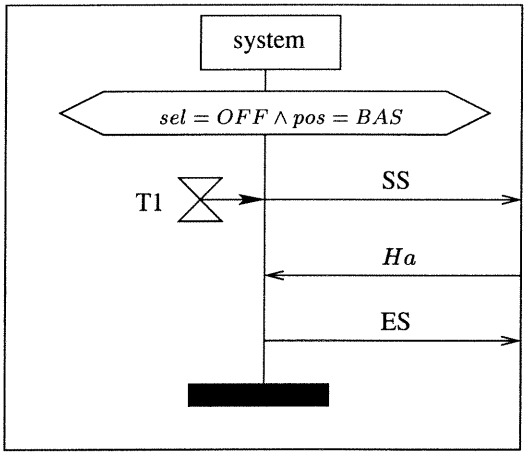
(a) Scénario d'un clic (sc_1)



(b) Scenario du clic simple (sc_2)



(c) Scénario d'un clic double (sc_3)



(d) Scénario de la sélection (sc_4)

FIG. 8.10 – Description du comportement du système de reconnaissance de click sous forme de MSCs

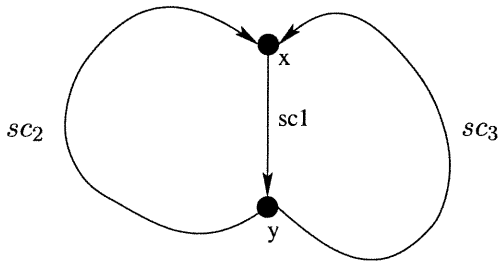


FIG. 8.11 – GIE de $Dir = \{sc_1 \rightsquigarrow sc_2, sc_2|sc_3, sc_2 \rightsquigarrow sc_1, sc_3 \rightsquigarrow sc_3\}$ du système de reconnaissance des clics d'une souris

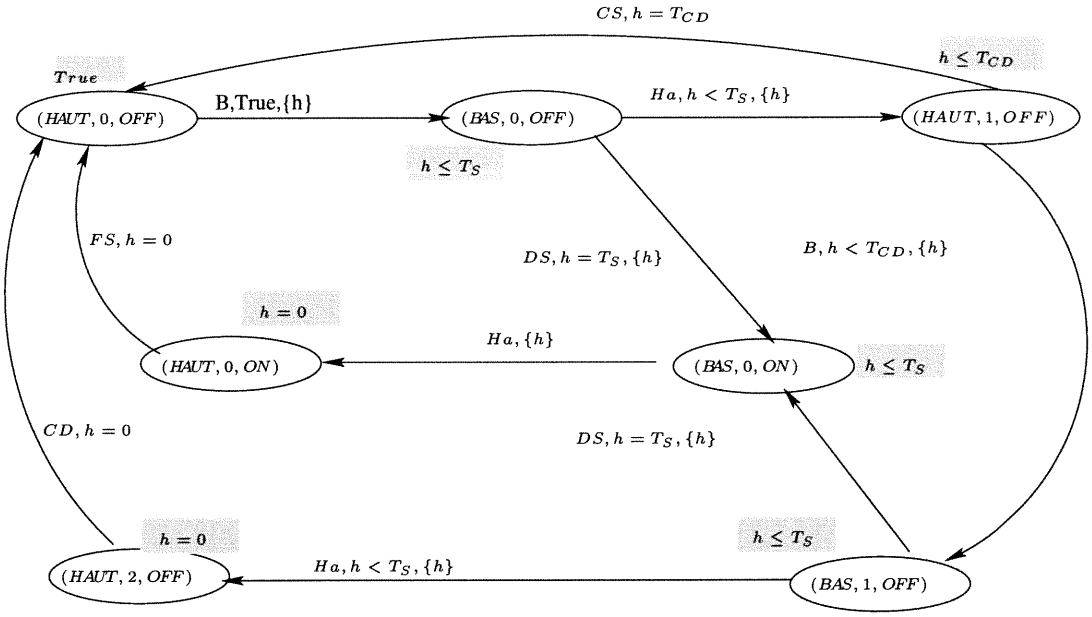


FIG. 8.12 – ATS résultant de l'intégration implicite libre des scénario sc'_1, sc'_2, sc'_3 et sc_4

sc	a	Description de l'action $sc.a$					
		$\varphi_{sc.a}$	$l_{sc.a}$	$\psi_{sc.a}$	$\varphi'_{sc.a}$	$\delta_{sc.a}$	$\lambda_{sc.a}$
sc_1	a_{00}	$True$	B	$sel{=}OFF$	$True$	$pos{:=}BAS$	$\{h\}$
	a_{10}	$h \leq T_S$	Ha	$True$	$h < T_S$	$pos{:=}HAUT$, $click{:=}\ click{+}1$	$\{h\}$
sc_2	a_{00}	$h \leq T_S$	CS	$True$	$h = T_S$	$click{:=}0$	$\{h\}$
sc_3	a_{00}	$h \leq T_{DC}$	B	$True$	$h < T_{DC}$	$pos{:=}BAS$	$\{h\}$
	a_{10}	$h \leq T_S$	Ha	$True$	$h < T_S$	$pos{:=}HAUT$, $click{:=}\ click{+}1$	$\{h\}$
	a_{20}	$h = 0$	CD	$True$	$h = 0$	$click{:=}0$	$\emptyset$
sc_4	a_{00}	$True$	DS	$pos{=}BAS, sel{=}OFF$	$h = T_S$	$click{:=}1$, $sel{:=}ON$	$\{h\}$
	a_{10}	$h \leq T_S$	Ha	$True$	$True$	$pos{:=}HAUT$, $click{:=}0$	$\{h\}$
	a_{20}	$h = 0$	FS	$True$	$h = 0$	$sel{=}OFF$	$\{h\}$

TAB. 8.1 – Description des scénarios sc_1, sc_2, sc_3 and sc_4

particuliers avant de proposer une méthode de résolution du cas général basée sur le GIE et l'algorithme de propagation. Ce qui permet d'aboutir, s'il existe, à un ensemble de scénarios qui satisfont les directives d'intégration explicite.

En spécifiant les directives d'intégration explicite, le concepteur peut déterminer de manière automatique les pré-conditions nécessaires aux scénarios contraints par ces directives. Ensuite ces scénarios, ainsi que d'autres peuvent être intégrés en utilisant les modes d'intégration implicite libre, Légo ou bien mixte.

Les deux chapitres précédents ont offert des algorithmes pour trouver les pré-conditions de certains scénarios et pour les intégrer en un automate temporisé sécuritaire. Le chapitre suivant présente un outil implémentant ces algorithmes et qui constitue l'ébauche d'une méthode de conception incrémentale basée sur notre approche scénario.

Chapitre 9

D'un Outil vers une méthodologie de spécification

Une continuation naturelle des résultats que nous avons présentés dans les chapitres précédents est le développement d'un outil d'aide à la spécification implémentant notre approche et que nous avons baptisé S_CE_NA. Un premier prototype de S_CE_NA a été développé en CAML, un langage fonctionnel qui permet une rapidité de développement. La dernière version de S_CE_NA est développée en C++. Le choix de C++ réside dans ses qualités de rapidité d'exécution, de facilité de programmation et sa facilité d'extension grâce à une conception orientée objet.

Dans ce chapitre, la section 9.1 présente les principales fonctionnalités de l'outil S_CE_NA et donne un aperçu sur son utilisation à travers la description de son interface. La section 9.2 met l'accent les aspects que S_CE_NA offre pour l'aide à la vérification de la spécification courante du système. La section 9.3 présente deux exemples de sessions d'utilisation de S_CE_NA.

9.1 Présentation de l'outil S_CE_NA

9.1.1 Les possibilités de S_CE_NA

Nous avons donné une description détaillée de l'utilisation de S_CE_NA dans un manuel de l'utilisateur [SBKD01]. Nous décrivons ici sommairement les possibilités de S_CE_NA comme suit :

- l'acquisition de la description du domaine et la vérification syntaxique de sa cohérence.
- l'acquisition de la spécification des scénarios et leur vérification syntaxique par rapport à la description du domaine d'application courant.

- la vérification de la consistance des scénarios acquis.
- l'acquisition d'un ensemble de directives d'intégration, la vérification syntaxique de ces directives et la résolution du PSED par la construction du GIE.
- la construction d'un prototype du système sous forme d'un ATS obtenu :
  - soit par le mode d'intégration implicite mixte des scénarios qui englobent aussi les modes d'intégration implicite libre et Légo
  - soit par le mode d'intégration explicite des scénarios
- la réduction de l'ATS résultant de l'intégration des scénarios.
- la retraçabilité dans les deux sens, entre les éléments d'un ATS résultant de l'intégration des scénarios et les actions des ces scénarios. Pour chaque transition ou place d'un ATS qui est générée par S_CE_NA nous pouvons en retracer le ou les scénarios d'origines et inversement. Ainsi si la vérification de cet ATS révèle une incohérence, nous pouvons déterminer les scénarios qui en sont la cause.

9.1.2 Interface de S_CE_NA

L'interface de S_CE_NA est sous forme d'un ensemble de menus textuels de choix. Le menu principal de S_CE_NA est représenté à la figure 9.1. Chaque choix dans le menu principal donne lieu à un sous-menu qui donne accès à ses activités. Nous décrivons pour chaque sous-menu du menu principal les activités essentielles qu'il permet.

- Sous-menu "`Gestion du domaine d'application`":
  Il offre la possibilité de charger un fichier où sont donnés les éléments formant la description du domaine d'application. Un exemple d'un tel fichier est représenté à la figure 9.2. La syntaxe du fichier de description du domaine d'application est donnée à l'annexe A. Le mot clé `clocks` permet la déclaration de l'ensemble des horloges H. `label` représente l'ensemble Ev. Le mot clé `domain` permet de déclarer une variable discrète de V et de donner son domaine de valeurs.
- Sous-menu "`Gestion des Scénarios et des Directives`":
  Ses activités sont l'acquisition des scénarios à partir de fichiers fournis par l'utilisateur, puis la vérification de leur cohérence syntaxique vis à vis de la description courante du domaine d'application. Comme exemple de fichiers de description des scénarios, nous avons représenté à la figure 9.3 sous, le format de S_CE_NA , le scénario donné par la table 7.1. Si un élément de description d'une action d'un scénario n'est pas décrit alors il correspond par défaut à $True$ si c'est une contrainte et à l'ensemble vide dans le cas d'une assigna-

```
.................. Menu Principal ................

Gestion du domaine d'application ..............  1
Gestion des Scénarios et des Directives........  2
Gestion des Actions Règles ....................  3
Gestion des Places ............................  4
Gestion des Transitions .......................  5
Changer le chemin des fichiers ................  6
Generation rapide de l'automate ...............  7
Quitter .......................................  0


Taper votre choix ....................>
```

FIG. 9.1 – Menu principal de l'outil S_CE_NA

```
Commutateur_telephonique
begin
clocks = {h1, h2}
labels = {pickup_A, send_tone_A, dialing_B, busy_tone_A,
          ring_A_B,pickup_B}
domain (A_status)={BUSY, IDLE}
domain (A_signal)={NONE, TONE, BUSY_TONE, DIALING(B),
                   ECHO_RING(B), TALKING(B)}
domain (B_status)={BUSY, IDLE}
domain (B_signal)={NONE, TONE, BUSY_TONE, RING(A),
                   TALKING(A)}
end
```

FIG. 9.2 – Fichier de description du domaine d'application dans le cas d'un commutateur téléphonique

tion d'horloges ou d'une affectation de variables. La syntaxe du fichier de description d'un scénarios est donnée à l'annexe B.

L'acquisition des directives d'intégration explicite est effectuée en chargeant un fichier de la forme représentée par la figure 9.4. Le chargement des directives n'affiche succès que si tous les scénarios qui interviennent dans ces directives ont déjà été lus.

Les activités de ce sous-menu peuvent être invoquées à n'importe quel moment lors d'une session de S_CE_NA , ainsi plusieurs fichiers de scénarios et de directives peuvent être chargés au besoin.

```
/* Fichier phone.sce */
begin_scenario Sc3_phone
   begin_node n0
      begin_action a01
         label: pickup_A
         variable_constraint:
           A_status = IDLE
           and A_signal = NONE
         variables_assignment:
           A_status := BUSY
         clocks_reset: {h2}
      end_action
   end_node
   begin_node n1
      begin_action a10
         invariant: h2 = 0
         label: send_tone_A
         temporal_guard: h2 = 0
         variable_assignment:
           A_signal := TONE
         clocks_reset: {h1}
      end_action
   end_node
   begin_node n2
      begin_action a20
         invariant: h1 <= 30
         label: dialing_B
```

```
/* Suite fichier phone_.sce*/
         temporal_guard: h1 < 30
         variable_assignment:
           A_signal := DIALING(B)
         clocks_reset: {h2}
      end_action
      begin_action a21
         invariant: h1 <= 40
         label: busy_tone_A
         temporal_guard: h1 = 30
         variable_assignment:
           A_signal := BUSY_TONE
      end_action
   end_node
   begin_node
      begin_action
         invariant: h2 = 0
         label: busy_tone_A
         variable_constraint:
           B_status = BUSY
         temporal_guard: h2 = 0
         variable_assignment:
           A_signal:= BUSY_TONE
         clocks_reset: {h2}
      end_action
   end_node
end_scenario
```

FIG. 9.3 – Exemple de fichier de description d'un scénario.

– Sous-menu "`Gestion des Actions Règles`":

```
/* Fichier : phone.dir */
Commutateur_telephonique
begin
sc_1 -> sc_2
sc_connect ->sc_2
Sc3_phone | sc_1
end
```

FIG. 9.4 – Exemple de description des directives d'intégration explicite

Ces activités permettent la génération des actions-règles des scénarios et vérifier leur consistance. La structure de donnée d'une action-règle comporte en plus des éléments qui la décrivent[1], la liste des actions avec leurs scénarios d'origine qui ont donné lieu à cette action-règle.

- Sous-menu "`Gestion des Places`":
  Il traite la détermination de la partition initiale et la réduction de l'ATS courant résultant de l'ensemble des actions-règles courant. Cette réduction est effectuée en invoquant la procédure ω-*minimisation*. Les places de l'automate quotient calculées peuvent être sauvegardées dans un fichier de sortie. La description de chaque place comporte :
  - un numéro de séquence comme identificateur,
  - la liste des actions-règles qui ont donné lieu à cette place. Nous distinguons les actions-règles pour lesquelles cette place est une place de départ et celles pour lesquelles elle est une place d'arrivée. Pour chaque action-règle nous indiquons l'action du scénario d'origine.
  - une contrainte de variables qui définit la classe de cette place, elle est représentée sous deux représentations équivalentes : sous forme d'un vecteur binaire et sous forme d'un ensemble d'interprétations de variables qui sont regroupées dans la classe de cette place.
  - et une contrainte d'horloges représentant l'invariant de cette place.

  Exemple 9.1. *Description d'une place*
  Un exemple d'une entrée dans le fichier décrivant les places de l'ATS quotient généré par S_CE_NA est donné comme suit :
  ---

[1] de la forme $(\psi_r, \varphi_r, l_r, \varphi'_r, \delta_r, \lambda_r)$

```
C20:
2 : [0,2,3] ; 5 : [1,4,1] ;
3 : [4,2,1] ; 0 : [0,1,2] ; 1 : [1,2,1] ;
00010001 00010
(NO,ON,0) (BAS,ON,0) (NO,ON,1)
h1<20
-------------------------------------------------
```

`C20` *est le nom affecté séquentiellement à cette place de l'ATS par* S_CE_NA.

`2 : [0,2,3]` *indique que cette place est une place de départ de l'action-règle dont l'indice est* `2` *dans la table des actions-règles de* S_CE_NA. *De plus,* `[0,2,3]` *indique que cette action-règle provient du scénario* `0` *et plus précisément de l'action* `3` *du noeud principal* `2`. *De même,* `5 : [1,4,1]` *indique la classe C20 est aussi place de départ l'action-règle 5.*

La ligne "`3 : [4,2,1] ; 0 : [0,1,2] ; 1 : [1,2,1] ;`*" est interprétée de manière identique celle qui la précède sauf que cette fois* `C20` *est une place d'arrivée des actions-règles* `3`, `0` *et* `1`.

`0001000100010` *est une représentation binaire d'une contrainte de variables qui définit la classe de* `C20`.

`(NO,ON,0) (BAS,ON,0) (NO,ON,1)` *sont les interprétations de variables qui satisfont la contrainte de variables qui définit la place* `C20`.

`h1<20` *représente l'invariant temporel de* `C20`.

La syntaxe générale du fichier de description des places est donnée à l'annexe D.

- Sous-menu "`Gestion des Transitions`":

Celui-ci permet la génération des transitions de ATS-EAR quotient. Dans le fichier décrivant les transitions, chaque entrée comporte les identificateurs des places de départ et d'arrivée, un garde temporelle et une assignation d'horloges ainsi qu'un champ pointant sur la liste des actions-règles qui sont l'origine de cette transition. Un exemple de description de transition est donnée par :

Exemple 9.2. *Description d'une transition*

```
T10 : 4 : [0,1,0] ;
C20--(Ha;(h<10);{h})-->C6
```

`T10` *représente un identificateur affecté de manière séquentiel à cette transition.*

`4 : [0,1,0]` *indique que cette transition provient l'action-règle d'indice 4 dans la table*

des actions-règles de S_CE_NA et `[0,1,0]` *représente les coordonnés d'une action d'un scénario d'où provient cette action-règle comme nous l'avons déjà mentionné pour les places de l'ATS-EAR quotient.*

`C20--(Ha ; (h<10) ;h)-->C6` *représente la description de la transition avec sa place de départ* `C20`*, son étiquette de synchronisation* `Ha`*, son garde temporel* `(h<10)`*, son assignation d'horloges* {h} *et sa place d'arrivée* `C6`.

La syntaxe générale du fichier de description des transitions est donnée à l'annexe D.

- Sous-menu : `Génération rapide de l'automate`

  Le choix de ce sous-menu permet la génération l'ATS-EAR quotient à partir des actions-règles courantes sans afficher les résultats intermédiaires.

9.2 Vers une méthodologie de spécification

Notre approche de génération de spécification à partir des scénarios permet d'établir une méthode de spécification incrémentale et itérative. ses principales activités sont la spécification de scénarios, la validation et la vérification. L'activité de vérification est plus générale que les propriétés que nous traitons dans l'outil S_CE_NA. Cependant, S_CE_NA permet la détection de certaines incohérences et oublis dans la spécification à l'aide du :

(a) contrôle de la syntaxe et de l'exécutabilité des scénarios. C'est que nous avons appelé *consistance* d'un scénario (définition 6.4).

(b) détection du chevauchement entre les scénarios ou du non déterminisme

(c) détection d'incomplétude de spécification

Le point (a) étant traité au chapitre 6 section 6.2.2, nous ne développons que les points (b) et (c).

9.2.1 Détection de chevauchement

Le chevauchement entre scénarios est illustré à la figure 7.1. Il est souvent l'effet du mode d'intégration implicite libre. L'existence du chevauchement n'est pas toujours une anomalie car il peut permettre de combler un oubli dans la spécification. Cependant, la découverte du chevauchement indésirable permet de détecter des interactions entre certaines fonctionnalités du système. Il est d'usage que la conception d'une fonctionnalité d'un système réactif soit pratiquée séparément du reste des autres fonctionnalités du système. Ceci correspond dans notre approche à la spécification de cette fonctionnalité sous forme d'un scénario par exemple. La vérification du scénario en tant qu'entité isolée peut ne montrer aucune anomalie. Par contre, lorsque ce scénario est intégré

dans le prototype courant du système, le nouveau prototype résultant n'est pas cohérent à cause à de l'interaction entre la nouvelle fonctionnalité et les anciennes. La détection de l'incohérence revient à déterminer les chevauchements indésirables.

La détection de chevauchements entre scénarios consiste à parcourir l'ATS résultant de leur intégration pour déterminer toutes les places qui proviennent de plus d'un scénario. Le concepteur peut juger qu'un chevauchement en une place de l'ATS est indésirable, si au cours du processus de validation, l'une des traces de l'exécution du système qui passe par cette place est incorrecte. Dans ce cas, il faut éliminer ce chevauchement en utilisant au besoin les opérations suivantes :

1. étendre le domaine d'une variable ou bien ajouter de nouvelles variables
2. ajouter de nouvelles horloges
3. ajouter de nouveaux événements
4. modifier un scénario existant
5. ajouter un nouveau scénario à la spécification
6. changer le mode d'intégration d'un scénario
7. ajouter de nouvelles directives d'intégration explicite

9.2.2 Détection du non-déterminisme

Le non déterminisme est un cas particulier du chevauchement entre les scénarios. Pour le détecter nous avons besoin d'une information supplémentaire sur les actions des scénarios. Le concepteur doit spécifier pour chaque action d'un scénario si elle est contrôlable ou pas. Les actions contrôlables sont celles pour lesquelles le système prend la décision de les exécuter. Un exemple d'action contrôlable est l'envoi d'un message. Les actions non contrôlables sont subis par le système de la part de son environnement. Par exemple, la réception d'un message n'est pas contrôlable pour le système.

Nous distinguons deux types de non-déterminisme :

- Le premier est dit non déterminisme au sens du test (figure 2.2(a)). Il est observé lorsqu'il existe un état dans un STE à partir duquel sortent deux transitions étiquetées par le même événement et qui mènent à deux états d'arrivée distincts.
- Le second type de non déterminisme est présent lorsque deux transitions partent d'un même état et sont respectivement étiquetées par :

- soit un événement contrôlable et un événement non contrôlable
- soit par deux événements contrôlables

Ces types de non déterminisme peuvent être détectés dans l'ATS formant le prototype du système. Grâce à la capacité de retraçabilité de S_CE_NA, nous pouvons remonter aux scénarios en cause pour éliminer cette incohérence.

9.2.3 Complétude de la spécification

Une spécification est complète si elle présente une description exhaustive de tous les comportements possibles du système. Décider de la complétude d'une spécification est un problème dit indécidable, c'est-à-dire, qui ne possède pas une solution algorithmique : cela ne veut pas dire que ce problème est insoluble comme on le croit parfois, mais qu'il est intrinsèquement difficile et qu'il requiert une intervention humaine [MS97]. De manière générale, il est difficile de décider de la complétude de la spécification du système car la spécification de tout système peut être étendue par de nouveaux comportements. Ces comportements peuvent être soit un oubli dans la spécification courante soit ils sont déduits à partir de nouvelles exigences visant une extension du système. Ainsi la difficulté de décider de la complétude de la spécification est un problème classique qui n'est pas dû à notre approche de génération de spécification par intégration de scénarios.

Cependant, dans notre approche le concepteur peut éviter l'oubli de comportements qui sont décrits dans les exigences des utilisateurs s'il s'assure pour chacun des aspects du comportement du système qu'il y a au moins un scénario qui le traite.

Par ailleurs, nous pensons que si l'ATS du prototype du système est non connexe, alors il est probable, mais non certain, que la spécification soit incomplète. Pour s'assurer que la non connexité n'est pas due à l'incomplétude, il faut déterminer pourquoi l'ATS du prototype du système est-il non connexe.

9.3 Exemples de sessions d'utilisation de S_CE_NA

Dans cette section nous décrivons deux sessions d'utilisation de S_CE_NA dans les lesquelles nous présentons comment certaines incohérences sont détectées. Puis nous proposons une alternative pour les corriger.

9.3.1 Cas d'un système avec non déterminisme

Nous reprenons l'exemple du système de reconnaissance des clics de la souris qui est présenté à l'exemple 2 de la section 7.2.3 afin d'illustrer la détection et la correction du non déterminisme en utilisant S_CE_NA. La description du domaine d'application est donnée à la figure 7.5. Le fichier décrivant le domaine d'application dans S_CE_NA est représenté à la figure 9.5.

```
      \*MOUSE_CLICK : Fichier mc.var*\
begin
clocks = {h}
labels = {B, Ha, CS, CD, DS, FS}
domain (pos)={HAUT, BAS}
domain (click)={O,N}
domain (sel)={ON, OFF}
end
```

FIG. 9.5 – Fichier de description du domaine d'application pour le système de reconnaissance de clics de souris

Le comportement du système est spécifié par les scénarios sc_1, sc_2, sc_3 et sc_4 de la table 7.2 dont le fichier qui les décrit est représenté dans l'annexe E. L'ATS résultant de l'intégration implicite libre de tous ces scénarios est représenté par la figure 7.6. En parcourant cet ATS pour la recherche de chevauchement entre les scénarios nous avons trouvé que la place $(Haut, O, OFF)$ résulte des scénarios sc_1 et sc_3. Le chevauchement entre sc_1 et sc_3 cause un non déterminisme car la place $(Haut, O, OFF)$ comporte deux transitions contrôlables qui correspondent respectivement aux événements SC (clic simple) et CD (clic double). Ce qui est incorrect car l'événement CD ne peut être envoyé qu'après deux clics consécutifs séparés par un délai inférieur à T_{CD}. Pour éliminer ce non déterminisme il faut pouvoir distinguer la place $(Haut, O, OFF)$ qui résulte de sc_2 de celle qui résulte du scénario sc_3. En analysant les scénarios sc_2 et sc_3, nous pouvons conclure que le non déterminisme est causé par le fait que l'état du système ne mémorise pas le nombre de clics. Il s'avère que pour mémoriser le nombre de clics, la variable *click* devrait avoir trois valeurs possibles et non pas deux. Donc nous étendons le domaine de la variable *click* pour devenir $dom(click) = \{N, O, OO\}$. La valeur N représente que le système n'a encore observé aucun clic. La valeur O représente que le système a déjà observé un clic qui est modélisé par l'événement B suivi de Ha. La valeur OO représente que le système a déjà observé deux clics.

Après avoir observé deux clics, à ce moment ci, le système pourra générer l'événement CD du scénario sc_3. Ainsi, nous modifions la spécification du scénario sc_3 pour obtenir le scénario sc'_3 qui est décrit par la table 9.1.

a	Description de l' action $sc'_3.a$					
	$\varphi_{sc'_3.a}$	$l_{sc'_3.a}$	$\psi_{sc'_3.a}$	$\varphi'_{sc'_3.a}$	$\delta_{sc'_3.a}$	$\lambda_{sc'_3.a}$
a_{00}	$h \leq T_{CD}$	B	$pos = HAUT \wedge click = O$ $\wedge sel = OFF$	$h < T_{DC}$	pos:=BAS	$\{h\}$
a_{10}	$h \leq T_S$	Ha	$True$	$h < T_S$	pos:=$HAUT$, $click := OO$	$\{h\}$
a_{20}	$h = 0$	DC	$True$	$h = 0$	$click$:=N	$\emptyset$

TAB. 9.1 – Description du scénario sc'_3 qui résulte de la modification du scénario sc_3 de la table 7.2

Le résultat généré par S_CE_NA lors de l'intégration implicite libre des scénarios sc_1, sc_2, sc'_3 et sc_4 est l'ATS décrit par les fichiers de l'annexe F. Nous avons dessiné cet ATS à la figure 9.6. Nous remarquons que le non déterminisme de l'ATS de la figure 7.6 a été éliminé dans l'ATS de la figure 9.6 grâce à l'élimination du chevauchement entre les scénarios sc_2 et sc'_3. Cependant l'ajout de la valeur OO pour la variable $click$ rend possible l'exécution du scénario sc_4 à partir de la place (BAS, OO, OFF), ce qui est incorrect. Afin d'écarter cette possibilité nous modifions la précondition de sc_4 en sur-contraignant sa première action principale par la contrainte $(click = N \vee click = O)$. La modification du scénario sc_4 donne lieu au scénario sc'_4 décrit par le tableau 9.2. L'ATS de l'intégration des scénarios sc_1, sc_2, sc'_3 et sc'_4 est dessiné à la figure 9.7.

a	Description de l' action $sc'_4.a$					
	$\varphi_{sc'_4.a}$	$l_{sc'_4.a}$	$\psi_{sc'_4.a}$	$\varphi'_{sc'_4.a}$	$\delta_{sc'_4.a}$	$\lambda_{sc'_4.a}$
a_{00}	$True$	DS	$pos = BAS \wedge sel = OFF \wedge$ $(click = N \vee click = O)$	$h = T_S$	$click$:=N, sel:=ON	$\{h\}$
a_{10}	$h \leq T_S$	Ha	$True$	$True$	pos:=$HAUT$	$\{h\}$
a_{20}	$h = 0$	FS	$True$	$h = 0$	sel:=OFF	$\{h\}$

TAB. 9.2 – Description du scénario sc'_4 qui résulte de la modification du scénario sc_4 de la table 7.2

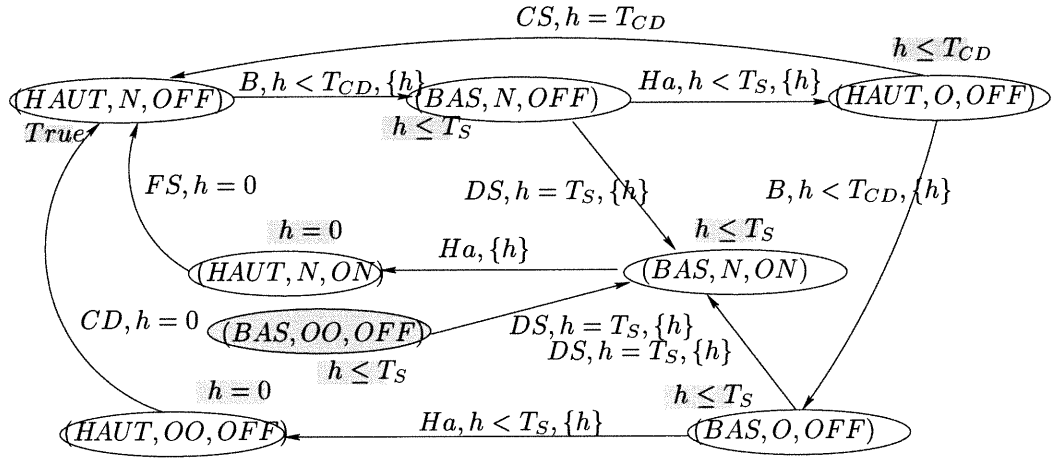


FIG. 9.6 – Graphe de l'automate temporisé sécuritaire obtenu par l'intégration implicite libre des scénarios sc_1, sc_2 et sc_4 de la table 7.2 et le scénario sc'_3 de la table 9.1. Chaque place de l'automate est caractérisée par un triplet de valeurs de la forme $(\omega(pos), \omega(click), \omega(sel))$ et une contrainte d'horloges comme invariant.

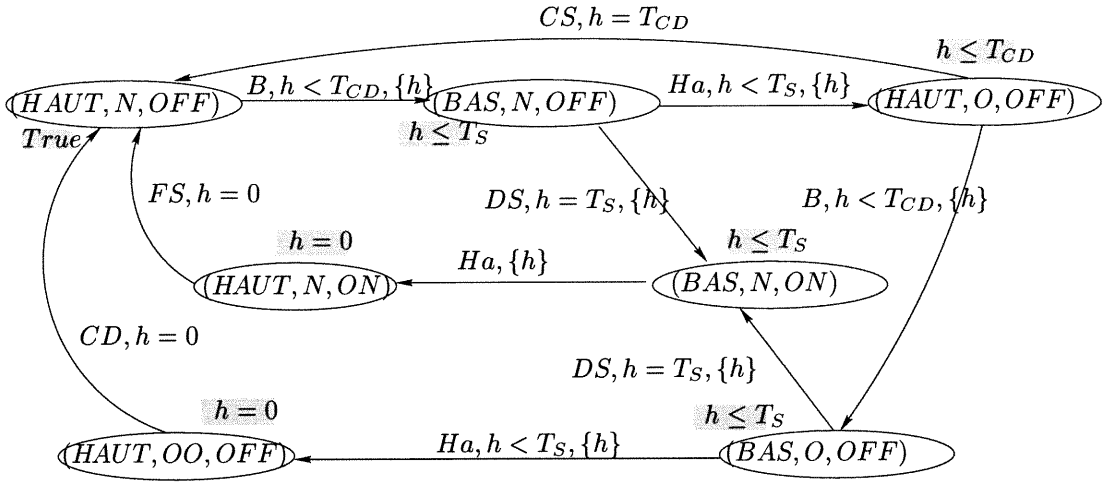


FIG. 9.7 – Graphe de l'automate temporisé sécuritaire obtenu par l'intégration implicite libre des scénarios sc_1, sc_2 sc_4 de la table 7.2 et le scénario sc'_3 de la table 9.1 et sc'_4 de la table 9.2.

9.3.2 Cas d'un système avec possibilité de blocage

Pour cette sous-section, nous adoptons la spécification du système de reconnaissance de clic, décrite dans la section 7.4.3. Nous considérons l'ATS de la figure 7.9 qui résulte de l'intégration implicite mixte des scénarios $\{sc'_1, sc'_2, \overline{sc'_3}, sc_4\}$ où sc'_1, sc'_2 et sc'_3 sont décrits par la table 7.3 et sc_4 par la table 7.2. Cet ATS présente à la place $(Haut, N, OFF, NO\_SC)$ un risque de blocage. En effet, le système peut rester indéfiniment dans cette place si l'événement B n'est pas observé pendant que la contrainte $h < T_{CD}$ est satisfaite. S_CE_NA permet de fournir la liste des scénarios d'où provient la place $(Haut, N, OFF, NO\_SC)$. En considérant ces scénarios, il s'est avéré que l'une des causes de ce blocage est que le scénario sc'_1 n'offre aucune alternative à l'action $sc'_1.a_{00}$.

L'une des solutions permettant d'éliminer la possibilité du blocage serait d'ajouter une action alternative dans le scénario sc'_1 pour l'action a_{00} et qui permettra d'initialiser l'horloge h dès que la contrainte $h < T_{CD}$ n'est pas satisfaite. Pour cela il faudrait ajouter un nouvel événement qui va étiqueter l'action alternative de $sc'_1.a_{00}$. Le nouvel événement peut être considéré comme un événement interne (notée ε) qui n'est pas observé par l'environnement du système. Ainsi, nous aurons $Ev = \{D, U, SC, DC, SS, ES, \varepsilon\}$. L'extension du scénario sc'_1 donne lieu à un nouveau scénario sc''_1 qui est décrit par la table 9.3.

a	Description de l'action a					
	$\varphi_{sc.a}$	$l_{sc.a}$	$\psi_{sc.a}$	$\varphi'_{sc.a}$	$\delta_{sc.a}$	$\lambda_{sc.a}$
a_{00}	$h \leq T_{CD}$	B	$pos = HAUT \wedge click = N \wedge sel = OFF$	$h < T_{CD}$	pos:=BAS	$\{h\}$
a_{01}	$h \leq T_{CD}$	ε	$True$	$h = T_{CD}$	$\emptyset$	$\{h\}$
a_{10}	$h \leq T_S$	Ha	$True$	$h < T_S$	pos:=$HAUT$, $click$:= O	$\{h\}$

TAB. 9.3 – Description du scénarios sc''_1

L'intégration implicite mixte des scénarios $\{sc''_1, sc'_2, \overline{sc'_3}, sc_4\}$ donne lieu à l'ATS de la figure 9.8. Nous remarquons que la transition étiquetée par $(\varepsilon, h = T_{CD}, \{h\})$ permet d'éliminer la possibilité de blocage dans la place $(Haut, N, OFF, NO\_SC)$ de cet ATS .

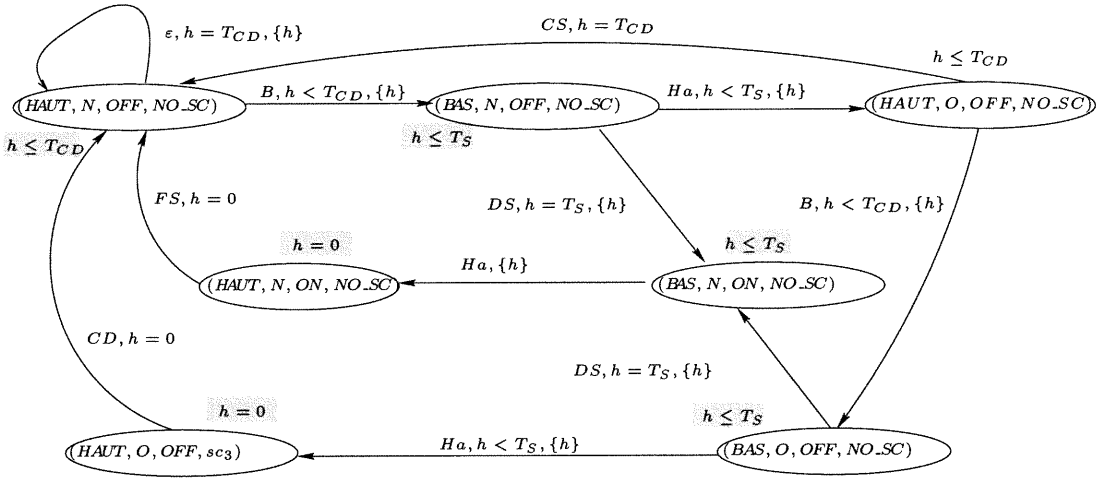


FIG. 9.8 – Graphe de l'automate temporisé sécuritaire obtenu par l'intégration implicite mixte de l'ensemble des scénarios $\{sc_1'', sc_2', \overline{sc_3'}, sc_4\}$. sc_1'' est décrit par la table 9.3 alors que sc_2', sc_3' et sc_4 sont spécifiés par la table 7.3

9.4 Conclusion

Nous avons présenté dans ce chapitre une description de l'outil S_CE_NA permettant l'intégration automatique de scénarios. Le langage de programmation C++ a été utilisé pour implanter l'outil. Ceci a permis le développement d'un outil extensible. L'outil S_CE_NA implante les modes d'intégration implicites libre, Légo et mixte. Il permet aussi le mode d'intégration explicite en implantant l'algorithme de résolution du problème de satisfaction d'un ensemble de directives d'intégration explicite. L'outil S_CE_NA représente le prototype d'un outil de support de synthèse de spécification basé sur l'intégration de scénario. S_CE_NA implante aussi certaines tâches de vérification du prototype du système comme la détection du chevauchement entre les scénarios, la détection du non déterminisme et de l'incohérence de certaines contraintes d'horloges ou de variables. En cas de détection d'une incohérence dans l'ATS du système, S_CE_NA permet de retracer les scénarios qui en sont la cause. L'outil S_CE_NA peut aussi être utilisé pour explorer la complétude de la spécification courante du système.

Chapitre 10

Conclusion

Dans cette thèse, nous avons développé une méthode de synthèse d'une spécification d'un système réactif temps réel en intégrant un ensemble de scénarios. La spécification résultante est modélisée sous forme d'un automate temporisé sécuritaire. Ce chapitre expose nos résultats et nos contributions et présente quelques extensions possibles à notre travail.

10.1 Contributions

Nous résumons dans cette section nos contributions majeures.

10.1.1 Modélisation et formalisation

Nous avons distingué dans le chapitre 5 deux aspects, statique et dynamique, dans la modélisation d'un système réactif. L'aspect statique représente la modélisation des caractéristiques du système en utilisant un ensemble de variables discrètes, un ensemble d'horloges à valeurs réelles et un ensemble d'étiquettes des événements. Les variables discrètes servent à concrétiser les propriétés non temporelles du système. Par exemple, si celui-ci comporte un interrupteur à deux positions alors nous pouvons associer à ce système une variable à deux valeurs. Les horloges permettent de chronométrer le temps écoulé depuis un événement donné et elle servent de référence afin de permettre ou d'interdire l'exécution des actions du système qui sont abstraites sous forme d'événements étiquetés comme cela est fait dans un STE. Nous nous sommes inspirés de la relation entre un ATS et son STE afin de définir une caractérisation de l'état d'un système qui est basée sur les

valeurs de ses variables discrètes et de ses horloges.

La modélisation de l'aspect dynamique du système représente la spécification de son comportement. Elle utilise les éléments de l'aspect statique comme éléments du langage pour décrire le comportement du système. Nous avons représenté l'unité élémentaire du comportement du système comme étant une action-règle qui décrit comment l'état du système change suite à l'exécution d'une action. Nous avons défini la sémantique d'un ensemble d'actions-règles par un STE auquel nous avons associé un ATS. Nous avons aussi proposé un algorithme pour minimiser l'ATS d'un ensemble d'actions-règles afin de réduire l'explosion combinatoire de sa taille.

L'un des piliers de cette thèse était de proposer au chapitre 6, un nouveau modèle convivial pour décrire les scénarios de comportement du système. Chaque scénario représente un ensemble de traces de l'exécution du système. L'originalité de ce modèle permet de cacher l'aspect formel de la description du comportement tout en tenant compte de sa dépendance par rapport au temps. La sémantique d'un scénario est reliée à celle d'un ensemble d'actions-règles qui représente sa forme canonique. Ainsi nous avons établi la relation entre notre modèle de scénario et les ATSs.

10.1.2 Synthèse d'un ATS par intégration de scénarios

L'intégration des scénarios consiste à les fusionner en un seul ATS. La caractérisation de l'état du système sert de ciment pour fusionner les scénarios. Elle permet la reconnaissance des comportements qui sont communs entre plusieurs scénarios. Nous avons distingué deux catégories de méthodes d'intégration des scénarios : l'intégration implicite au chapitre 7 et l'intégration explicite au chapitre 8. Pour chacune des méthodes nous avons prouvé la relation de conformité entre les scénarios d'origine et l'ATS résultant de leur intégration. La catégorie des méthodes d'intégration implicite indique que les scénarios sont fusionnés sans aucune contrainte supplémentaire sur leurs ordonnancements. Dans cette catégorie nous avons défini le mode d'intégration implicite libre et le mode d'intégration implicite Légo, puis la combinaison de ces deux modes a donné lieu au mode d'intégration implicite mixte. Le mode d'intégration implicite libre se distingue par sa fine granularité basée sur l'action d'un scénario. Il est utile pour la découverte des exigences des utilisateurs ainsi que pour repérer les interactions de services. Par contre, le mode d'intégration implicite Légo permet de protéger l'unité de chaque scénario dans la spécification générée. Dans ce mode, les scénarios représentent les blocs qui composent la spécification. L'exécution d'un scénario n'est pas interruptible car la granularité du mode d'intégration implicite Légo est le scénario.

La méthode intégration explicite est basée sur la spécification des directives d'intégration qui sont considérées comme des contraintes sur l'ordonnancement des scénarios. Nous avons proposé un algorithme pour la satisfaction des directives d'intégration. Les scénarios qui résultent de cette étape sont intégrés en utilisant les modes d'intégration implicite. Ceci constitue l'un des aspects par lequel notre approche se distingue car elle permet la combinaison des modes d'intégration implicite et explicite dans la même spécification.

10.1.3 Vérification

Nous avons traité la vérification de la spécification d'un système sur trois étapes :

Vérification syntaxique : La déclaration de la description du domaine d'application permet aux concepteurs de s'entendre sur une modélisation des propriétés du système. Ainsi nous pouvons assurer que toute propriété du système soit modélisée de manière unique pour tous les intervenants dans la conception. Tous les scénarios doivent respecter une syntaxe qui n'utilise que les éléments déclarés dans la description du domaine d'application. La première étape dans la vérification consiste de s'assurer que les scénarios sont syntaxiquement correctes et que les expressions de leurs contraintes sont satisfaisables.

Vérification du comportement d'un scénario : La deuxième étape de la vérification porte sur le comportement du scénario. Elle consiste à vérifier l'exécutabilité des actions-règles qui représentent la forme canonique du scénario. Cette étape de vérification ne garantit pas que le scénario est correct.

Vérification de l'ATS-EAR : La troisième étape de la vérification du système représente la vérification de l'ATS résultant de l'intégration des scénarios. Cette thèse n'a couvert que quelques aspects de la vérification de cet ATS. Les aspects que nous avons couverts concernent la détection du chevauchement des scénarios et la détection du non déterminisme. Nous avons aussi partiellement traité la détection de l'incomplétude de la spécification.

10.1.4 Développement d'un outil

Nous avons développé un outil (S_CE_NA) implémentant notre approche. Il représente un prototype d'un environnement de support à la spécification des systèmes réactifs temps-réels. L'outil S_CE_NA permet tous les modes d'intégration de scénarios et les tâches de vérification qui sont présentés dans cette thèse. Il permet aussi une retraçabilité dans les deux sens : des scénarios vers

l'ATS résultant de leur intégration et inversement. Ainsi en cas de détection d'une incohérence dans cette ATS, nous pouvons directement remonter aux actions des scénarios qui en sont la cause.

10.2 Quelques extensions possibles

Dans cette section, nous présentons quelque propositions comme extensions possibles de notre approche.

10.2.1 Extension du modèle cible

Il est possible d'étendre le modèle des automates cibles sous forme d'automates temporisés étendus avec les variables. Cette extension permet de traiter des systèmes où les domaines des variables sont infinis. Nous distinguons à ce moment deux types de variables

- des variables discrètes à domaines finis que nous pouvons appeler variables de place car elle permettent la définition des places de l'automate
- des variables à domaines éventuellement infinis. Ces variables sont appelées variables de contrôle. Elle sont des variables globales similaires aux variables associées à une FSM étendue par les variables.

Pour effectuer ces extensions, il faut étendre le modèle des scénarios utilisé dans *SCENA* afin de pouvoir alimenter le nouveau modèle cible de la spécification. Un exemple de spécification qui peut être obtenue par cette approche est donné à la figure 10.1. Les places de cet automate sont définies en fonction des variables de places par contre nous remarquons que ses transitions sont formées par une étiquette de synchronisation, une contrainte temporelle (garde), une contrainte sur les variables de contrôle, une initialisation d'horloges et un bloc d'instructions sur les variables de contrôle.

Une autre alternative d'extension du modèle cible serait de synthétiser un automate hybride [ACH+95, Hen96] à partir de scénarios. Pour cela il faudrait aussi changer le modèle des scénarios en conséquence afin de pourvoir alimenter la synthèse d'un automate hybride.

10.2.2 Synthèse d'un système à composantes parallèles ou concurrentes

Lorsqu'un système comporte plusieurs composantes, l'outil *SCENA* permet la synthèse d'une spécification du système à partir de scénarios de comportement. Cette spécification peut être vue

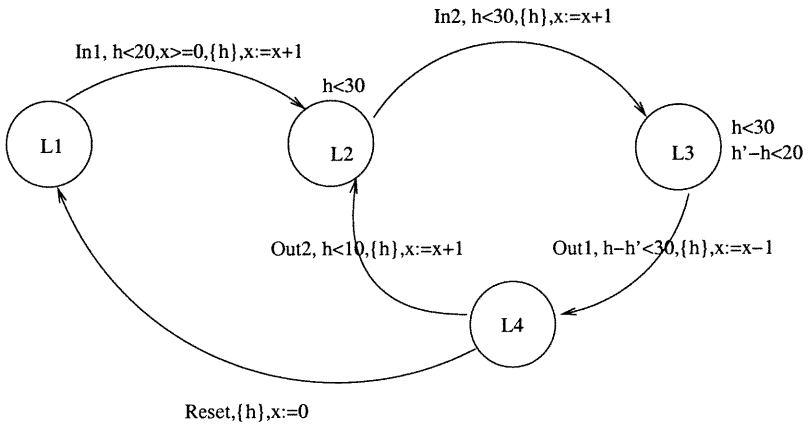


FIG. 10.1 – Exemple d'un automate temporisés sécuritaire étendu par les variables. h et $\mathcal{H}$ sont des horloge, x est une variable de contrôle.

comme une composition parallèle des spécifications des composantes du système. Cette approche ne permet pas d'avoir la spécification de chaque composante du système. Le besoin d'avoir une spécification pour chaque composante est motivé par des fins de réutilisation de ces spécifications.

Le problème que nous nous posons consiste à synthétiser la spécification d'un système réactif temps réel constitué de plusieurs composantes communicantes. Nous cherchons à obtenir la spécification de chaque composante. Pour synthétiser la spécification d'un tel système, nous distinguons deux approches possibles qui sont basées sur les scénarios.

La première approche consiste à utiliser l'outil S_CE_NA pour synthétiser chacune des composantes du système en supposant que cette composante représente le système à synthétiser. Ainsi, les composantes du système sont isolément synthétisées une à une. Ensuite, il faut définir la fonction de synchronisation [NSY92, LPY95] qui permet une correspondance entre des événements dans différentes composantes du système. La spécification du système regroupe la spécification de chacune de ses composantes et la fonction de synchronisation.

La seconde approche consiste à synthétiser simultanément les composantes du système à partir de scénarios concurrents. Pour cela il faut enrichir les scénarios de comportement, que nous utilisons dans S_CE_NA dans le but de pouvoir repérer les comportements communs à plusieurs composantes. Pour chaque action appartenant à de tels scénarios, soit elle est commune à plu-

sieurs composantes soit elle est propre à une seule composante. Une action commune à plusieurs composantes va se retrouver dans leurs spécifications respectives. Les actions propres à une composante sont soit des actions internes à la composante, soit des interactions de la composante avec l'environnement du système.

La seconde approche a l'avantage d'être plus efficace que la première pour les raisons suivantes :

- La deuxième approche traite simultanément toutes les composantes du système, ce qui lui offre une vision globale qui permet d'effectuer en même temps certaines tâches de vérification.
- L'approche synthétisant composante par composante souffre de beaucoup de répétitions ennuyeuses qui sont évitées par l'autre approche.

10.2.3 Méthodologie de l'ingénierie des exigences

Notre travail a donné lieu l'esquisse d'une méthode d'ingénierie des exigences qui permet la synthèse d'une spécification formelle du système. L'esquisse de notre méthode pourra être raffinée par un travail d'expérimentation et pourra être fusionnée avec des méthodologies plus générales telle que UML [UML97] UML.

Annexe A

Syntaxe de la description du domaine d'application

Elle permet de décrire les domaines des variables, l'ensemble des horloges et l'ensemble des étiquettes de synchronisation (les événements).

A.1 Description du domaine d'application

Extension du fichier : le fichier doit avoir l'extension ".var"

Mot-clés : `begin`, `clocks`, `labels`, `domain` et `end`. Tous les mots clés sont en minuscules.

Identificateur : c'est une suite des lettres minuscules ou majuscules, des chiffres et de caractère '_' . Un identificateur doit commencer par une lettre.

entier : est une suite des chiffres qui peuvent être préfixes par le signe '+' ou '-'.

Syntaxe :

```
<domain_description_file> ::=  [<titre_variable>]
                               begin <corps_variable> end
<titre_variable> ::= <texte>

<corps_variable> ::= <liste_clock> <liste_label>
                     <liste_variable>
```

```
<liste_clock> ::= clocks = "{" <clocks> "}"

<clocks> ::= <identificateur> [, <clocks>]

<liste_label> ::= labels = "{" <labels> "}"

<labels> ::= <identificateur>  [, <labels>]

variables ::= <une_variable> [<variables>]

<une_variable> ::= domain ( <identificateur> )
                   = { <liste_valeurs> }

liste_valeurs ::= <une_valeur> , [<liste_valeurs>]

une_valeur ::= <entier> |
               <identificateur> |
              "[" <entier> ":" <entier> "]"  |
               <attribute_val>

<attribute_val> ::= <identificateur> "(" <attribut_parametres> ")" |
               <identificateur> "(" ")"

<attribut_parametres> ::= <un_attribut_parametre>
                          [, <attribut_parametres>]


<un_attribut_parametre> ::= <entier> | <identificateur>
```

A.2 Fichier de description des attributs

Les attributs sont des valeurs composées que peuvent prendre les variables. Ils servent à la vérification des types dans les expressions des domaines des variables.

Extension du fichier : le fichier doit avoir l'extension ".att"

Mot-clés : `begin` et `end`

Syntaxe :

```
<Attribute_description_file> ::= [<titre_attribut>] <corps_attribut>
<titre_attribut> ::= <texte>
<corps_attribut> ::= begin <un_attribut>* end
<un_attribut> ::= <attribute_name>"("<liste_valeur_parametre>")"
                  | <attribute_name>  "(" ")"

<attribute_name>::= <identificateur>

<liste_valeur_parametre> ::= "{" <valeur_parametre>
                                  [,<liste_valeur_parametre> ] "}"
<valeur_parameter> ::=   <entier> |
                         <identificateur> |
                         "[" <entier> ":" <entier> "]"
```

Annexe B

Syntaxe des scénarios

Extension du fichier : le fichier doit avoir l'extension ".sce"

Mot-clés : `begin_scenario`, `end_scenario`, `begin_node`, `end_node`, `begin_action`, `end_action`, `invariant`, `label`, `variable_constraint`, `temporal guard`, `variable_assignment`, `clock_reset`, `true`, or, and. Tous les mots clés sont en minuscules.

Syntaxe :

```
scenario_description_file ::=  [<titre_scenario>] <scenario>*
<titre_scenario>::=<text>
<scenario> ::= begin_scenario <sc_id>
                                <liste_node>+ end_scenario
<sc_id> ::= <entier> | <identificateur>
<liste_node> ::= begin_node [<identificateur>]
                                        <une_action>+ end_node
<une_action> ::= begin_action [<identificateur>]
                    <action> end_action
<action> ::= [<invariant]> <label> [<variable_constraint>]
             [<temporal_guard]>] [<variable_assignment>]
             [<clock_reset>]
<invariant> ::= invariant ":" [<liste_clock_constraint>] |
                invariant ":" [true]
<label> ::= label ":" >identificateur>
```

```
<variable_constraint> ::= variable_constraint ":" [expression] |
                          variable_constraint ":" [true]
<temporal_guard> ::=
          temporal_guard ":" [<liste_clock_constraint>] |
          temporal_guard ":" [true]
<variable_assignment> ::= variable_assignment
                          ":" [<liste_assignment>]
<clocks_reset> ::= clocks_reset ":" ["{" [<liste_clock>] "}"]
<liste_clock> ::= <identificateur> ["," <liste_clock>]
<liste_clock_constraint> ::= <clock_constraint>
                             ["," <liste_clock_constraint>]
<clock_constraint> ::=
  <identificateur> <order_relation> <entier> |
  <identificateur> "-" <identificateur> <order_relation> <entier>
<order_relation> ::=  "<" | "<=" | ">" | ">=" | "="
<expression> ::= <expression> "or" <and_expression> |
               <and_expression>
<and_expression> ::= <and_expression> "and" <feuille>
                   <feuille>
<feuille> ::= "(" <expression> ")" |
            <identificateur> <var_relation> <entier> |
            <identificateur> "=" <identificateur> |
            <identificateur> "=" <attribute_val>
<var_relation> ::= "<" | ">" | "<=" | ">=" | "!=" | "="
<liste_assignation> ::= <une_assignation>
                        ["," <liste_assignation>]
<une_assignation> ::=
  <identificateur> ":=" <identificateur> |
  <identificateur> ":=" <entier> |
  <identificateur> ":=" <entier>"*"<identificateur>"+"<entier> |
  <identificateur> ":=" <entier>"*"<identificateur>"-"<entier> |
  <identificateur> ":=" <identificateur> "+" <entier> |
  <identificateur> ":=" <identificateur> "-" <entier> |
```

```
<identificateur> ":=" <entier> "*" <identificateur> |
<identificateur> ":=" <attribute_val>
```

Annexe C

Syntaxe des directives d'intégration explicite

Extension du fichier : le fichier doit avoir l'extension ".dir"

Mot-clés : `begin` et `end`. Tous les mots clés sont en minuscules.

Syntaxe :

```
directive_description_file ::= [<texte>]  <corps_directive>

<corps_directive> ::= begin <liste_directive> end

<liste_directive> ::= <une_directive> [<liste_directive>]

<une_directive> ::= <sc_id> "|" <sc_id> |
                    <sc_id> "->"  <sc_id>

<sc_id>::= <entier> | <identificateur>
```

Annexe D

Syntaxe des fichiers résultats décrivant l'ATS généré par $S_C E_N A$

Dans cet annexe nous decrivons les fichiers de sortie décrivant l'automate temporisé syntétisé par $S_C E_N A$. La description des cet ATS est composée de deux fichiers, l'un décrivant les places de l'automate et l'autre ses transitions.

D.1 Fichier décrivant les place de l'ATS

```
<Fichier_places>::= <une_place>*
 <une_place>::= C<id>:"\n"
                <AR_depart> * "\n"
                <AR_arrivée> * "\n"
                <vecteur_binaire>"\n"
                <Liste_interpretation_de_variables>"\n"
                <contrainte_temp_invariant> "\n"

<AR_depart> ::= <indices_AR_depart> ":"
                "["<Scenario_id>","<Noeud_id>","<Action_id>"]" ";"

<AR_arrivée> ::= <indices_AR_arrivée> ":"
                 "["<Scenario_id>","<Noeud_id>","<Action_id>"]" ";"
```

D.2 Ensemble des transitions

```
<Fichier_places> ::= <une_transition>*
<une_transition> ::= C<id> : <AR>+ "\n"
                     <Place_Depart> "---"
                     "("<label>","<contrainte_horloge>",
                         "<assignation_horloge>")"
                         "--->" <Place_arivee> "\n"

<AR> ::= <indices_AR> ":"
         "["<Scenario_id>","<Noeud_id>","<Action_id>"]" ";"
```

Annexe E

Fichier decrivant les scénarios du système de reconnaissance des clics d'une souris selon le formalisme de SCE_NA

Nous listons ci-dessous le fichier décrivant les scénarios sc_1, sc_2, sc_4 de la table 7.2 et le scénario sc'_3 de la table 9.1.

```
SCENARIOS_DES_CLICS_DE_SOURIS
begin_scenario sc1
  begin_node
    begin_action
       invariant: true
       label: B
       variable_constraint: pos=HAUT and
                            click=N  and
                            sel=OFF
       temporal_guard: true
       variable_assignment: pos:=BAS
       clock_reset: {h}
```

```
    end_action
  end_node
  begin_node
    begin_action
       invariant: h <= 4
       label: Ha
       variable_constraint: true
       temporal_guard: h < 4
       variable_assignment: pos:=HAUT ,
                            click:=0
       clock_reset: {h}
    end_action
  end_node
end_scenario

begin_scenario sc2
  begin_node
    begin_action
       invariant: h <= 4
       label: CS
       variable_constraint: pos=HAUT and
                            click=0  and
                            sel=OFF
       temporal_guard: h = 4
       variable_assignment: click:= N
       clock_reset: {h}
    end_action
  end_node
end_scenario

begin_scenario sc'3
  begin_node
    begin_action
```

```
        invariant: h <= 2
        label: B
        variable_constraint: pos=HAUT and
                             click=O and
                             sel=OFF
        temporal_guard: h < 2
        variable_assignment: pos:=BAS
        clock_reset: {h}
    end_action
  end_node
  begin_node
    begin_action
        invariant: h <= 4
        label: Ha
        temporal_guard: h < 4
        variable_assignment: pos:=HAUT ,
                             click:=OO
        clocks_reset: {h}
    end_action
  end_node
  begin_node
    begin_action
        invariant: h=0
        label: CD
        temporal_guard: h=0
        variable_assignment: click:=N
    end_action
  end_node
end_scenario

begin_scenario sc4
  begin_node
    begin_action
```

```
        label: DS
        variable_constraint: pos=DOWN and
                             sel=OFF

        temporal_guard: h=4
        variable_assignment: click:=N, sel:=ON
        clock_reset: {h}
    end_action
  end_node
  begin_node
    begin_action
        invariant: h<=4
        label: Ha
        variable_constraint: true
        temporal_guard: true
        variable_assignment: pos:=HAUT
        clock_reset: {h}
    end_action
  end_node
  begin_node
    begin_action
        invariant: h=0
        label: FS
        temporal_guard: h=0
        variable_assignment: sel:=OFF
        clock_reset: {h}
    end_action
  end_node
end_scenario
```

Annexe F

Description d'un automate temporisé généré par S_CE_NA

Nous listons ci-dessous les fichiers des places et des transitions générés par S_CE_NA et qui décrivent l'ATS résultant de l'intégration implicite libre des scénarios sc_1, sc_2, sc_4 de la table 7.2 et du scénario sc'_3 de la table 9.1.

F.1 Fichier des places

```
C0 :
0:[0,0,0];
2:[1,0,0]; 5:[2,2,0]; 8:[3,2,0];
00000100 0000
(HAUT,N,OFF)
{True}
C1 :
1:[0,1,0]; 6:[3,0,0];
0:[0,0,0];
00000000 0001
(BAS,N,OFF)
{True}
C2 :
```

```
2:[1,0,0]; 3:[2,0,0];
1:[0,1,0];
01000000 0000
(HAUT,O,OFF)
{(h<=4)}
C3 :
4:[2,1,0]; 6:[3,0,0];
3:[2,0,0];
00000001 0000
(BAS,O,OFF)
{True}
C4 :
5:[2,2,0];
4:[2,1,0];
00010000 0000
(HAUT,OO,OFF)
{(h==0)}
C5 :
6:[3,0,0];

00000000 0100
(BAS,OO,OFF)
{True}
C6 :
7:[3,1,0];
6:[3,0,0];
00000000 0010
(BAS,N,ON)
{(h<=4)}
C7 :
8:[3,2,0];
7:[3,1,0];
00001000 0000
```

```
(HAUT,N,ON)
{(h==0)}
```

F.2 Fichier des transitions

```
T0 : [0,0,0];
C0--(B;True;{h})-->C1
T1 : [0,1,0];
C1--(Ha;(h<4);{h})-->C2
T2 : [3,0,0];
C1--(DS;(h==4);{h})-->C6
T3 : [1,0,0];
C2--(CS;(h==4);{h})-->C0
T4 : [2,0,0];
C2--(B;(h<2);{h})-->C3
T5 : [2,1,0];
C3--(Ha;(h<4);{h})-->C4
T6 : [3,0,0];
C3--(DS;(h==4);{h})-->C6
T7 : [2,2,0];
C4--(CD;(h==0))-->C0
T8 : [3,0,0];
C5--(DS;(h==4);{h})-->C6
T9 : [3,1,0];
C6--(Ha;(h<=4);{h})-->C7
T10 : [3,2,0];
C7--(FS;(h==0);{h})-->C0
```

Bibliographie

[ACD93] R. Alur, C. Courcoubetis, and D. Dill. Model-checking in dense real-time. *Information and Computation*, 104(1) :2–34, mai 1993.

[ACH+92] R. Alur, C. Courcoubetis, N. Halbwachs, D. Dill, and H. Wong-Toi. Minimization of timed transition systems. *Lecture Notes in Computer Science*, 630 :340–354, 1992.

[ACH+95] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138(1) :3–34, février 1995.

[AD90] R. Alur and D. L. Dill. Automata for modeling real-time systems. In Michael S. Paterson, editor, *Automata, Languages and Programming, 17th International Colloquium*, volume 443 of *Lecture Notes in Computer Science*, pages 322–335, Warwick University, England, 16–20 juillet 1990. Springer-Verlag.

[AD94] R. Alur and D. Dill. A Theory of Timed Automata. *Theoretical Computer Science*, 126 :183–235, 1994.

[ALB97] D. Amyot, L. Logrippo, and R.J.A. Buhr. Spécification et conception de systèmes communicants : une approche rigoureuse basée sur des scénarios d'usage. In *CFIP97*, pages 159–174, Liège, Belgique, septembre 1997.

[All83] J. Allen. Maintaining Knowledge about Temporal Intervals. *Communications of the ACM*, 26(11) :832–843, novembre 1983.

[Alu99] R. Alur. Timed automata. *Lecture Notes in Computer Science*, 1633 :8–22, 1999.

[Arn92] Andre Arnold. *Systemes de transitions finis et sémantique des processus communicants*. Masson, 1992.

[BC95] R.J.A. Buhr and R.S. Casselman. *Use Case Maps for Object-Oriented System*. Prentice Hall, USA, 1995.

[BFH90] A. Bouajjani, J-C. Fernandez, and N. Halbwachs. Minimal model generation. In *Proc. 2nd International Computer Aided Verification Conference*, pages 197–203, 1990.

[BFH+92] A. Bouajjani, J.-C. Fernandez, N. Halbwachs, P. Raymond, and C. Ratel. Minimal state graph generation. *Science of Computer Programming*, 18, 1992.

[BHR84] S. D. Brookes, C. A. R. Hoare, and A. W. Roscoe. A theory of communicating sequential processes. *Journal of the ACM*, 31(3) :560–599, juillet 1984.

[BJR97] G. Booch and I. Jacobson J. Rumbaugh. *The Unified Modeling Language for Object-Oriented Development*. Rationale Software Corporation, Santa Clara, CA, documentation set version 1.0 edition, 1997.

[BK76] A. W. Biermann and R. Krishnaswamy. Constructing programs from example computations. *IEEE Trans. Software Engineering*, SE-2(9) :141–153, 1976.

[Bri88] E. Brinksma. A Theory for the Derivation of Tests. In S. Aggarwal and K. Sabnani, editors, *Proceedings of the 8th International Symposium on Protocol Specification, Testing and Verification*, pages 63–74. IFIP, North-Holland, 1988.

[Büc62] J. R. Büchi. On a decision method in restricted second order arithmetic. In *Proceedings of the International Congress on Logic, Method, and Philosophy of Science*, pages 1–12, Stanford, CA, USA, 1962. Stanford University Press.

[Buh98] Raymond J. A. Buhr. Use case maps as architectural entities for complex systems. *IEEE Transactions on Software Engineering*, pages 1131–1155, decembre 1998.

[Chr84] Philippe Chrétienne. *Les réseaux de Pétri temporisés*. Publications M.A.S.I., Paris : Universite P. et M. Curie, Institut de programmation : CNRS, ERA 592, 1984.

[CR83] J. E. Coholahan and N. Roussopoulos. Timed Requirements for Timed-Driven Systems using Augmented Petri Nets. *IEEE transactions on Communication Protocols*, 9 :603–616, 1983.

[Dan96] Bénédicte Dano. Spécifications dynamiques des besoins orientées objet : une démarche fondée sur les scénarios. In *Proceedings du XIVe Congres INFORSID*, pages 53–78, Bordeaux, France, juin 1996.

[Dan97] Bénédicte Dano. *Une démarche d'ingénierie des besoins orientée objet guidée par les cas d'utilisation*. PhD thesis, Université de Nantes, novembre 1997.

[DBB97] B. Dano, H. Briand, and F. Barbier. A use case driven requirements engineering process. *Requirements Engineering Journal, Springer Verlag*, 2(2) :79–91, 1997.

[dH84] R. de Nicola and M. C. B. Hennessy. Testing equivalences for processes (concurrent programming). *Theoretical Computer Science*, 34(1-2) :83–133, novembre 1984.

[Dil90] D. L. Dill. Timing assumptions and verification of finite-state concurrent systems. In J. Sifakis, editor, *Proceedings of the International Workshop on Automatic Verification Methods for Finite State Systems*, volume 407 of *LNCS*, pages 197–212, Berlin, juin 1990. Springer.

[DKFM97] J. Desharnais, R. Khedri, M. Frappier, and A. Mili. Integration of sequential scenarios. *Lecture Notes in Computer Science*, 1301 :310–326, 1997.

[Dri00] Jawad Drissi. *Vers la construction automatique d'un module inconnu dans un système composé.* PhD thesis, Université de Montréal, Août 2000.

[DSVS99] R. Dssouli, S. Some, J. Vaucher, and A. Salah. Service creation environment based on scenarios. *Information and Software Technology*, 41(11-12) :697–713, 1999.

[DY96] C. Daws and S. Yovine. Reducing the number of clock variables of timed automata. In *Proceedings of the 17th IEEE Real-Time Systems Symposium (RSS '96*, pages 73–81, Washington - Brussels - Tokyo, decembre 1996. IEEE.

[EK98] M. Elkoutbi and R. K. Keller. Modeling Interactive Systems with Hierarchical Colored Petri Nets. In *"Proceedings of the 1998 Advanced Simulation Technologies Conference"*, pages 432–437, Boston, MA, avril 1998.

[EMSS92] E. A. Emerson, A. K. Mok, A. P. Sistla, and J. Srinivasan. Quantitative Temporal Reasoning. *Journal of Real-Time Systems*, 4 :331–352, 1992.

[Fer90] Jean-Claude Fernandez. An implementation of an efficient algorithm for bisimulation equivalence. *Science of Computer Programming*, 13(2–3) :219–236, mai 1990.

[Fur93] Ulrich Furbach. Formal Specification Methods for Reactive Systems. *J. Systems Software*, 21 :129–139, 1993.

[Gar89] H. Garavel. *Compilation et vérification de programmes LOTOS.* PhD thesis, Université Joseph Fourier – Grenoble I, novembre 1989.

[Gil62] A. Gill. *Introduction to the theory of finite-state machines.* Mc Graw-Hill, New-York, 1962.

[Gli95] M. Glinz. An integrated formal model of scenarios based on statecharts. In W. Schäfer and P. Botella, editors, *Proceedings of the Fifth European Software Engineering Conference*, number 989 in Lecture Notes in Computer Science, pages 254–271. Springer-Verlag, septembre 1995.

[GMMP91] C. Ghezzi, D. Mandrioli, S. Morasca, and M. Pezze. A Unified High-Level Petri Net Formalism for Time-Critical Systems. *IEEE transactions on Communication Protocols*, 17 :160–172, 1991.

[GR85] U. Goltz and W. Reisig. CSP–programs as nets with individual tokens. In G. Rozenberg, editor, *Advances in Petri Nets 1984*, volume 188 of *Lecture Notes in Computer Science*, pages 169–196. springer, 1985.

[Har87] David Harel. STATECHARTS : A Visual Formalism for Complex Systems. *Science of Computer Programming*, 8 :231–274, 1987.

[Hen85] M. Hennessy. Acceptance trees. *Journal of the ACM*, 32(4) :896–928, octobre 1985.

[Hen96] Thomas A. Henzinger. The theory of hybrid automata. In *Proceedings, 11th Annual IEEE Symposium on Logic in Computer Science*, pages 278–292, New Brunswick, New Jersey, 27–30 juillet 1996. IEEE Computer Society Press.

[Hil89] H. P. Hillion. Timed Petri nets and application to multi-stage production systems. In Springer-Verlagr, editor, *Lecture Notes in Computer Science, Advances in Petri Nets 1989*, pages 281–305, Berlin - Heidelberg - New York, juin 1989.

[HLN+88] D. Harel, H. Lachover, A. Naamad, A. Pnueli, M. Politi, R. Sherman, and A. Shtul-Trauring. STATEMATE ; a working environment for the development of complex reactive systems. In *Proceedings of the 10th International Conference on Software Engineering*, pages 396–406, Singapore, avril 1988. IEEE Computer Society Press.

[HN96] David Harel and Amnon Naamad. The STATEMATE Semantics of Satecharts. *ACM Transactions on Software Engineering and Methodology*, 5(4) :293–333, octobre 1996.

[HNSY94] Thomas A. Henzinger, Xavier Nicollin, Joseph Sifakis, and Sergio Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111(2) :193–244, juin 1994.

[Hoa85] C.A.R Hoare. *Communicating Sequential Processes.* Prentice-Hall, 1985.

[HSG+94] P. Hsia, J. Samuel, J. Gao, D. Kung, Y. Toyoshima, and C. Chen. Formal approach to scenario analysis. *IEEE Software*, 11 :33–41, mars 1994.

[HU79] J. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Language, and Computation.* Addison–Wesley, Reading, MA, 1979.

[ISO88] ISO8807 : Information processing systems — open systems interconnection : LOTOS – a formal description technique based on the temporal ordering of observational behaviour, 1988. (ISO International Standard 8807).

[Jen92] Kurt Jensen. *Coloured Petri nets : basic concepts, analysis, methods, and practical use.* EATCS Monographs on Theoretical Computer Science. Springer-Verlag, Berlin ; New York, 1992.

[KL94] Inhye Kang and Insup Lee. State minimization for concurrent system analysis based on state space exploration. In *Compass'94 : 9th Annual Conference on Computer Assurance*, pages 123–134, Gaithersburg, MD, 1994. National Institute of Standards and Technology.

[KM94] K. Koskimies and E. Mäkinen. Automatic Synthesis of State Machines from Trace Diagrams. *Software-Practice and Experience*, 24(7) :643–658, juillet 1994.

[Koh78] Zvi Kohavi. *Switching and Finite Automata Theory.* McGraw-Hill Book Company, New York, 1978.

[KS83] Paris C. Kanellakis and Scott A. Smolka. CCS expressions, finite state processes, and three problems of equivalence. In *Proceedings of the Second Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, pages 228–240, Montreal, Quebec, Canada, 17–19 Août 1983.

[LK91] C.A. Lakos and C.D. Keen. Simulation with Object-Oriented Petri Nets. In *Australian Software Engineering Conference.* Dept. of Computer Science, University of Tasmania, Australia, 1991.

[LKP93] C. A. Lakos, C. D. Keen, and E. J. Palmer. A flexible distributed simulator for object-oriented Petri nets. In J. Hulskamp and D. Jones, editors, *Transputers and Parallel Applications*, pages 141–146, Amsterdam, 1993. IOS Press.

[LPY95] Kim G. Larsen, Paul Pettersson, and Wang Yi. Compositional and symbolic model-checking of real-time systems. In *Proceedings of the 16th IEEE Real–Time Systems Symposium, Pisa, Italy*, pages 76–87, decembre 1995.

[LT88] Nancy Lynch and Mark Tuttle. An introduction to Input/Output Automata. Technical Report MIT/LCS/TM-373, MIT, Cambridge, Massachusetts, novembre 1988.

[Lus97] F. Lustman. A Formal Approach to Scenario Integration. *Annals of Software Engineering*, 3 :255–272, septembre 1997.

[MBC+95] M. A. Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. *Modelling with Generalized Stochastic Petri Nets*. John Wiley and Sons, 1995.

[Mil80] R. Milner. *A Calculus for Communicating Processes*, volume 92 of *Lecture Notes in Computer Science*. Springer Verlag, 1980.

[Mil89] Robin Milner. Communicating and concurrency. In Prentice-Hall, editor, *Lecture Notes in Computer Science*, 1989.

[MS97] J. Monin and J. Sifakis. *Elément de classification des méthodes formelles*. Observatoire Français des Techniques Avancées, Masson Ed, 1997.

[NSY92] X. Nicollin, J. Sifakis, and S. Yovine. Compiling real-time specifications into extended automata. *IEEE Transactions on Software Engineering*, 18(9) :794–804, septembre 1992. Special Issue on the Specification and Analysis of Real-Time Systems.

[Ost93] Jonathan S. Ostroff. Visual Tools for Verifying Real-Time Systems. In *First AMAST International Workshop on Real-Time Systems*, novembre 1993.

[Par81] David Park. Concurrency and automata on infinite sequences. In *Theoretical Computer Science, 5th GI-Conf.*, LNCS 104, pages 167–183. Springer-Verlag, Karlsruhe, mars 1981.

[Pha94] Marc Phalippou. *Relations d'implantation et hypothèses de test sur des automates à entrées et sorties*. PhD thesis, Université de Bordeaux I, septembre 1994.

[PT87] Robert Paige and Robert E. Tarjan. Three partition refinement algorithms. *SIAM Journal on Computing*, 16(6) :973–989, decembre 1987.

[RBP+91] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object-Oriented Modelling and Design*. Prentice-Hall Inc., 1991. ISBN 0-13-630054-5.

[SBKD01] Aziz Salah, Yassir Bellout, Dhafer Ben Khedher, and Rachida Dssouli. *Manuel de l'utilisateur de S_CE_NA*. Université de Montréal, septembre 2001.

[SD96] S. Somé and R. Dssouli. An Enhancement of Timed Automata generation from Timed Scenarios using Grouped States. Technical Report 1029, DIRO-Université de Montréal, 1996.

[SDL01a] Aziz Salah, Rachida Dssouli, and Guy Lapalme. Compiling real-time scenarios into a timed automaton. In Myungchul Kim, Byoungmoon Chin, and Sungwon Kang,

editors, *Formal Description Techniques and Protocol Specification, Testing and Verification FORTE XIV/PSTV XXI 2001*, pages 135–150, Cheju Island, Korea, Août 2001. Kluwer Academic Publishers.

[SDL01b] Aziz Salah, Rachida Dssouli, and Guy Lapalme. How to satisfy explicit integration directives to compile real time scenarios into a timed automaton. Technical Report 1209, Université de Montréal, septembre 2001.

[SDLV00] Aziz Salah, Rachida Dssouli, Guy Lapalme, and Daniel Vincent. Vers un environnement de création de services fondé sur les scénarios enrichis. In *Proceedings of Colloque Francophone sur l'Ingénierie des Protocoles*, pages 199–214, 2000.

[SDV95] S. Somé, R. Dssouli, and J. Vaucher. From Scenarios to Timed Automata : Building Specifications from Users Requirements. In *Proceedings of the 2nd Asia Pacific Software Engineering Conference (APSEC'95)*, pages 48–57. IEEE, decembre 1995.

[SDV96a] S. Somé, R. Dssouli, and J. Vaucher. Toward an automation of requirements engineering using scenarios. *Journal of Computing and Information*, 2(1) :1110–1132, 1996. Special issue : ICCI'96, 8th International Conference of Computing and Information, Waterloo, Ontario Canada.

[SDV96b] S. Somé, R. Dssouli, and J. Vaucher. Un cadre pour l'ingénierie des exigences avec des scénarios. In *CFIP'96 Colloque Francophone sur l'Ingenierie des Protocoles*, pages 187–206, octobre 1996.

[Sha92] Alan C. Shaw. Communicating real-time state machines. *IEEE Transactions on Software Engineering*, 18(9) :805–816, septembre 1992.

[Som97] S. T. Somé. *Un cadre d'ingénierie des exigences par scénarios d'interaction*. Phd thesis, Université de Montréal, juin 1997.

[Spi88] J. M. Spivey. *Understanding Z : A Specification Language and Its Formal Semantics*. Cambridge Tracts in Theoret. Cambrige University Press, UK, 1988.

[UML97] UML. Rational-Software-Corporation, Microsoft, Hewlett-Packard, Oracle, Sterling, MCI, Unisys, ICON, Intellicorp and i-Logix Rational Software Corporation, Santa Clara, CA, UMl notation guide, version 1.1, 1997.

[Yov93] S. Yovine. *Méthodes et outils pour la vérification symbolique de systèmes temporisés*. PhD thesis, Institut National Polytechnique de Grenoble, France, mai 1993.