**MASTER EN SCIENCES INFORMATIQUES**

# De l'utilisation de scenarios dans le contexte du langage Albert

Bongartz, Christophe

*Award date:*
1997

*Awarding institution:*
Universite de Namur

Facultés Universitaires Notre-Dame de la Paix à Namur

Institut d'Informatique

Année académique 1996-1997

# DE L'UTILISATION DE SCENARIOS DANS LE CONTEXTE DU LANGAGE ALBERT

Mémoire réalisé par **Christophe Bongartz**

en vue de l'obtention du diplôme

de Maître en Informatique

Promoteur : **Eric Dubois**

# DE L'UTILISATION DE SCENARIOS DANS LE CONTEXTE DU LANGAGE ALBERT

**Christophe Bongartz**

## RESUME

L'exploitation de scénarios dans l'ingénierie des besoins connaît actuellement un succès important. Les scénarios sont entre autre utilisés pour l'acquisition et la validation de spécifications. ALBERT est un langage de spécification formel conçu pour exprimer les besoins de systèmes composites temps-réel. Des recherches sont menées pour développer un animateur de spécifications ALBERT. L'objectif de ce mémoire est de construire un langage d'expression de scénarios adapté aux spécifications ALBERT et à l'utilisation en rapport avec l'animateur. Le langage est constitué de 3 parties : une première permettant d'exprimer le contenu d'un scénario et basée sur le formalisme des MSC, une deuxième permettant d'exprimer des pré- et postconditions de scénarios basée sur la logique temporelle des prédicats du premier ordre et une troisième permettant de structurer des informations liées à la construction des scénarios basée sur une approche informelle.

## ABSTRACT

The use of scenarios in the domain of requirements engineering is a success at this time. They are used among other things for acquisition and validation of specifications. ALBERT is a formal specification language designed for capturing requirements of composite real-time systems. Research work is done to develop an animator of specifications written in ALBERT. The objective of this thesis is to build a language for expressing scenarios adjusted to ALBERT specifications and to the use with an animation tool. The language has 3 parts: the first one allows us to represent the contents of a scenario based on the MSC formalism, the second one allows us to represent pre- and postconditions of scenarios based on first order temporal logic and the third one allows us to structure the information related to the construction of scenarios based on a informal approach.

Mémoire de Maîtrise en Informatique

Septembre 1997

Promoteur : Eric Dubois

*Je tiens à remercier vivement mon promoteur, le professeur Eric Dubois. Sans ses conseils, ses recommandations et ses indications judicieuses concernant le chemin à suivre, ce mémoire n'aurait jamais pu aboutir.*

*Merci aussi à Patrick Heymans qui était toujours disponible pour me fournir les explications et la documentation nécessaire.*

*Enfin, merci à Anne-Christine Groux qui a consacré beaucoup de temps à relire ce travail et à corriger mes erreurs de français.*

# Table des matières

# 1. Introduction

## 1.1 Motivation

Le cycle de vie du développement d'un logiciel passe par différentes phases. On identifie généralement une ou des phases d'analyse, de design, d'implémentation, de test et de déploiement. Le but d'une telle découpe du travail est d'arriver à construire des logiciels ayant entre autres les propriétés suivantes : répondre à un besoin d'un client, faciles à maintenir, sans erreur, réalisés en respectant les coûts et délais spécifiés, efficaces, réutilisables, etc.

On remarque actuellement que les premières phases de ce modèle, c'est-à-dire les tâches liées à l'analyse, deviennent de plus en plus importantes. Des études montrent par exemple que les erreurs diminuant les qualités d'un logiciel citées ci-dessus sont les plus importantes dans les premières phases du développement (cfr. [MDLMA97]). C'est dans ces premières phases que nous essayons de voir ce que veut le client. Mais comment pouvons-nous être certain que ce que nous avons compris est bien ce que le client veut ?

L'évolution des langages de programmation souligne cette tendance d'attribuer de plus en plus d'importance aux phases permettant de comprendre et de spécifier correctement les besoins des clients. Ils se rapprochent de plus en plus des langages exprimant des concepts liés au domaine du client. On trouve des langages de design orienté objet (OO) permettant d'exprimer des concepts sous forme d'objets du domaine du client et pouvant générer du code exécutable à partir de cela. Le pourcentage du temps que passe un informaticien à réfléchir à des problèmes techniques diminue.

L'ingénierie des besoins (IB) est la discipline essayant de trouver des méthodes, des outils et des modèles qui aident l'informaticien à comprendre et à spécifier correctement les besoins du client. Ainsi différentes techniques issues de réflexions théoriques et pragmatiques ont vu le jour. Une de ces techniques est l'exploitation de scénarios.

La réintroduction des « use case » par Ivar Jacobson [Jacobson 92] a déclenché une tendance nouvelle : la spécification par scénarios. Les « use case » connaissent un succès appréciable, malgré des fondements peu théoriques et encore moins formels. Certains auteurs essayent de les formaliser et d'apporter les fondements théoriques nécessaires mais l'exploitation des scénarios reste un domaine de recherche passionnant.

## 1.2 Objectifs

Le but principal de ce mémoire est de spécifier un langage d'expression et de structuration de scénarios dans le cadre d'une spécification en ALBERT.

ALBERT étant un langage formel essentiellement déclaratif, un langage d'expression de scénarios peut apporter une dimension complémentaire très intéressante.

Une première partie de ce langage se rapporte au contenu d'un scénario. Celui-ci est surtout caractérisé par l'expression d'une suite d'actions et l'échange de messages entre différents sujets. Les « message sequence charts » (MSC) définissent un langage graphique permettant d'exprimer des scénarios dans un environnement fortement coopératif. Moyennant certaines adaptations, les MSC permettent de refléter une spécification en ALBERT.

Une deuxième partie se rapporte au contexte d'un scénario. La bonne compréhension du contexte et l'expression des éléments importants aident à structurer des scénarios et à pouvoir les exploiter de manière efficace. Une bonne gestion des contextes permet également d'augmenter la réutilisation de scénarios.

Cet objectif s'inscrit aussi dans le cadre des recherches liées à un animateur du langage ALBERT [DDD 94] [Heymans]. En effet, un animateur permet d'expérimenter et de tester certaines caractéristiques d'une spécification de besoins. Le client se place dans un certain contexte et explique la façon dont le logiciel intégré dans son environnement devrait fonctionner. Il fait cela en décrivant des scénarios acceptables ou non acceptables.

Afin de mieux comprendre les idées exprimées dans ce mémoire, nous allons les appliquer à un cas concret. Nous avons choisi un système de gestion de feux rouges pouvant être placé à presque n'importe quel carrefour.

## 1.3 Plan de travail

Le mémoire est divisé en 3 grandes parties.

Premièrement, nous expliquerons de manière plus détaillée la manière dont différents auteurs comprennent l'IB et ce qui leur paraît intéressant (section 2.2). Nous préciserons ensuite notre compréhension de l'IB (sections 2.3et 2.4). Ensuite, nous allons énumérer quelques problèmes rencontrés dans l'ingénierie des besoins (sections 2.5 et 2.6). Nous spécifierons également l'étude de cas (section 2.7) et profiterons de l'occasion pour faire un petit rappel sur le langage ALBERT (section 2.8).

Deuxièmement, nous parlerons des scénarios en général, c'est-à-dire indépendamment d'une spécification et d'une animation ALBERT. Dans un premier temps, nous poserons quelques définitions (section 3.2) et nous rappellerons les utilités des scénarios (section 3.3). Ensuite, nous rentrons un peu plus dans le détails concernant la représentation (section 3.4) et le contenu (section 3.5) des scénarios avant d'identifier les problèmes liés à leur construction (section 3.6). Nous terminerons le 3ème chapitre par quelques exemples de scénarios trouvés dans la littérature (section 3.7).

Le 4ème et dernier chapitre avant la conclusion rentrera dans le vif du sujet. Nous expliquerons d'abord la différence entre le contenu et le contexte d'un scénario (section 4.1). Après un état de l'art dans le domaine de la représentation des scénarios (section 4.2), nous allons spécifier un langage à 3 parties permettant de représenter des scénarios dans le cadre du langage ALBERT (section 4.3). Nous terminerons par un exemple complet (section 4.3.6) basé sur l'étude de cas.

# 2. L'ingénierie des besoins (IB)

Après quelques définitions trouvées dans la littérature (section 2.1) nous allons expliquer notre compréhension de la discipline qu'est l'IB (section 2.2). Ensuite, nous allons détailler un peu plus le processus (section 2.3) et le résultat de l'IB (section 2.4). Les sections 2.5 et 2.6 identifient quelques problèmes intrinsèques de cette discipline et donnent des pistes de solutions à ces problèmes. Nous terminerons ce chapitre par un petit rappel concernant le langage ALBERT (section 2.7).

## 2.1 Introduction

Dans la littérature, nous trouvons plusieurs définitions différentes de l'IB. Tout d'abord, [Loucopoulos 95] définit l'IB comme suit :

"the systematic process of developing requirements through an iterative co-operative process of analyzing the problem, documenting the resulting observations in a variety of representation formats, and checking the accuracy of understanding gained"

Ensuite, [Hofmann 93] cite 3 références définissant l'IB :

« Boehm [Boehm 1979] defines RE as *the discipline for developing a complete, consistent, unambiguous specification – which can serve as a basis for common agreement among all parties concerned – describing what the software product will do (but not how it will do it ; this is to be done in the design specification).*

In the STARTS guide [STARTS Guide 1987] *RE comprises those processes by which the purchaser's statement of intention and requirement, written or spoken, are transformed into a precise, unambiguous, consistent and complete specification of system behaviour including functions, interfaces, performance and constraints.*

For Leite [Leite 1987] *requirements engineering is defined as a process in which 'what is to be done' is elicited and modeled. This process has to deal with different viewpoints, and it uses a combination of methods, tools, and actors. The product of this process is a model, from which a requirements document is produced.* »[1]

---

[1] RE est l'abréviation de « requirements engineering »

[Boehm 1979] : Boehm, B.W. (1979) : « Guidelines for Verifying and Validating Software Requirements and Design Specifications ». In : Sarnet, P.A. (ed) : IFIP 19979, North-Holland : Amsterdam. 711-719

[STARTS Guide 1987] : « Requirements Definition and Design ». In : The STARTS Guide : A guide to methods and software tools for the construction of large real-time systems, NCC Publications : Manchester. 177-223.

Enfin, dans son cours de méthodologie de développement de logiciel, Eric Dubois explique le rôle de l'ingénierie des besoins grâce au dessin suivant :

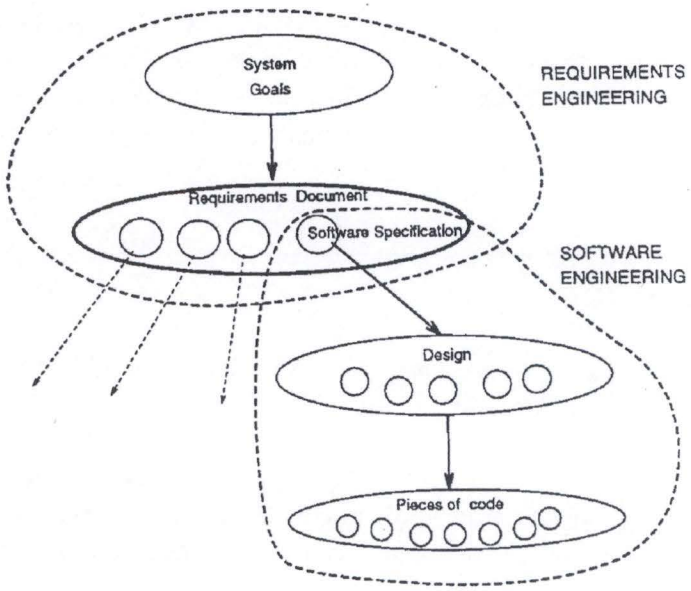



**Figure 1 - La portée de l'ingénierie des besoins**

Il distingue la partie ingénierie des besoins (requirements engineering) et la partie ingénierie du logiciel (software engineering). La première part des buts ultimes du client et produit un cahier des charges (requirements document). Certaines parties de ce document servent alors comme point de départ pour l'ingénierie du logiciel, matériel ou organisationnel.

Les 3 sections suivantes donnent une vue plus détaillée de ce que nous entendons par l'ingénierie des besoins.

## 2.2 Définitions

Le terme ingénierie reflète l'idée d'un processus systématique et préparé ainsi que l'idée d'innovation et de créativité. Presque chaque définition trouvée dans la littérature parle d'un processus. Il est donc important d'identifier et de décrire une démarche qui permet de trouver et d'exprimer les besoins d'un client et cela en faisant preuve de créativité. Dans la section 2.3 nous détaillerons les différents éléments de ce processus.

Un autre point commun aux différentes définitions données ci-dessus est le fait que le processus mène à un résultat documenté. Certains parlent d'un « requirements document »

---

[Leite 1987] : Leite, J.C. (1987) : « A Survey on Requirements Analyses ». Technical Report RTP-071, Department of Information and Computer Science, University of California at Irvine.

(cfr. [Hofmann 93] et [MDL96]) ou encore cahier des charges. Dans la section 2.4, nous détaillerons le contenu d'un tel cahier des charges.

Une autre notion importante est celle de besoin. Un besoin correspond à une sensation de manque et/ou d'insatisfaction par rapport à une situation existante. L'IB essaye de découvrir ce besoin chez le client et de trouver un consensus sur la solution permettant de satisfaire ce besoin. Plusieurs auteurs insistent sur le fait que le résultat de l'IB devrait exprimer le 'quoi' de la solution et non le 'comment'. Cependant, la distinction entre le 'quoi' et le 'comment' n'est pas toujours facile à faire. En plus, le besoin du client peut très bien se rapporter à un 'comment'. Il peut très bien dire qu'il a besoin d'un moyen efficace d'archivage de fax et ajouter qu'il aimerait bien que cet archivage se fasse de manière digitale et grâce à une base de donnée relationnelle. L'IB doit pouvoir découvrir et exprimer cela malgré le 'comment' déjà formulé.

L'ingénierie des besoins est donc le processus permettant d'arriver à l'expression d'une sensation de manque et/ou d'insatisfaction d'un client. Cette expression doit être admise par les parties impliquées dans le processus (client et fournisseur).

## 2.3 Le processus

[Loucopoulos 95] nous donne un cadre général pour déterminer le processus menant à l'expression des besoins (p.21). Il se focalise sur 3 axes principaux : l'élicitation, la spécification et la validation (cfr. Figure 2)

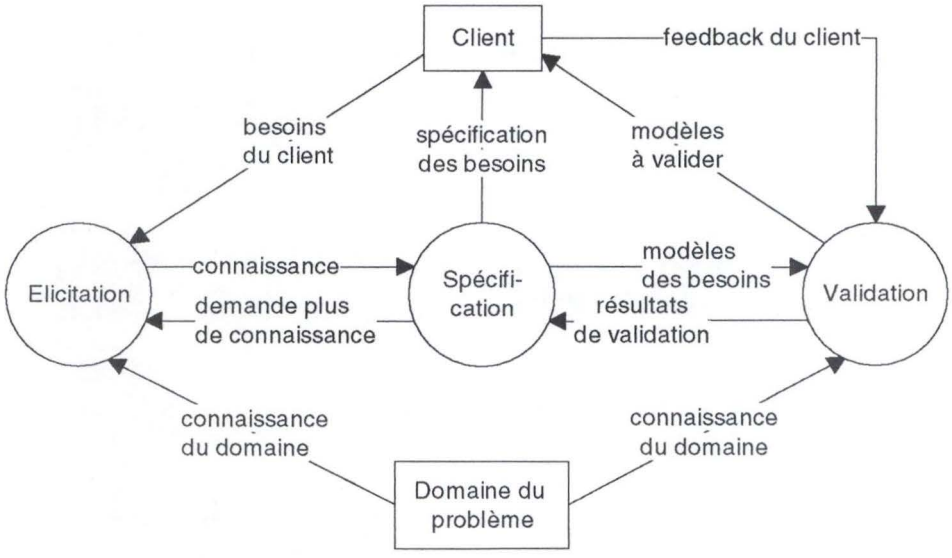



**Figure 2 – le processus de l'IB selon [Loucopoulos 95]**

- L'élicitation rassemble toutes les techniques permettant à l'analyste de comprendre les besoins du client. Elle consiste à appliquer différentes techniques permettant de récolter toutes les informations nécessaires à la bonne compréhension du domaine du

problème. Les sources d'information sont, par exemple, les experts du domaine, les systèmes existants ou encore la littérature concernant le domaine.

- La spécification consiste en des techniques d'expression des besoins formulés de manière acceptable pour toutes les parties intervenant dans le processus. Elle peut être vue comme une structuration des connaissances acquises lors de l'élicitation.

- La validation a pour but de vérifier ou de falsifier la compréhension de l'analyste. L'analyste ayant exprimé de manière structurée ce qu'il a compris, le client peut voir si cela correspond à ce qu'il a voulu dire.

Ces trois phases s'exécutent plus ou moins en parallèle. Un analyste fera un peu d'élicitation suivi de spécification et de validation pour une petite partie du problème. Il recommence alors ce cycle afin d'améliorer et d'augmenter sa compréhension.

Dans [Dubois 95] ces différentes phases sont structurées différemment (cfr. Figure 3).

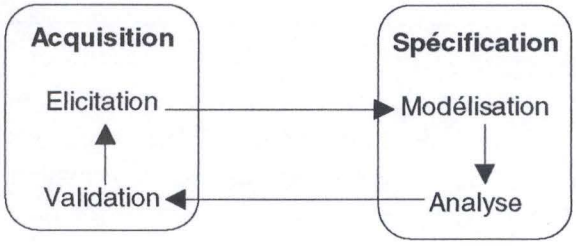



**Figure 3 - phases de l'IB selon [Dubois 95]**

La phase de spécification est divisée en deux parties : la modélisation et l'analyse.

- La modélisation correspond à la spécification de [Loucopoulos 95].

- L'analyse consiste en la détection par l'analyste d'incohérences ou de contradictions dans les modèles déjà élaborés.

L'élicitation et la validation font partie d'une activité d'acquisition. L'acquisition met l'accent sur l'interaction entre l'analyste et le client tandis que la spécification est un travail réalisé par l'analyste uniquement.

ALBERT se focalise surtout sur la partie spécification tandis que ce mémoire se focalise surtout sur la partie acquisition (élicitation et validation). C'est dans ce sens que le mémoire apporte un élément complémentaire au langage ALBERT.

## 2.4 Le résultat

Le résultat de l'ingénierie des besoins est un document (au sens large) exprimant de manière satisfaisante les besoins du client pour toutes les parties intéressées. Ce document est

appelé cahier des charges. Il sera exprimé sous différentes formes équivalentes en fonction des utilisations et en fonction des personnes devant le comprendre.

Voici quelques besoins de formes d'expression du résultat : à partir du cahier des charges,

- un gestionnaire de projet doit pouvoir estimer le temps de travail nécessaire à la réalisation d'une solution respectant ce cahier des charges,
- dans le cadre d'une solution software, un analyste doit pouvoir déterminer les propriétés essentielles de la solution,
- un utilisateur doit pouvoir valider la manière dont il aimerait bien utiliser la solution,
- un juge doit pouvoir décider si une solution satisfait les besoins exprimés ou non,
- etc.

Vu que le cahier des charges doit être compréhensible par ces différentes personnes et vu qu'il doit servir de base pour un contrat entre client et fournisseur, il faut qu'il ait les caractéristiques suivantes, caractéristiques qu'on trouve dans [Dubru 93].

- Cohérence et complétude : il est rare qu'un analyste n'interroge qu'un seul client pour connaître les besoins. Il est souvent amené à devoir consolider des avis contradictoires et complémentaires de plusieurs personnes. Un cahier des charges est cohérent si on n'y trouve aucune contradiction et il est complet si tous les éléments qui s'y trouvent ont été bien définis. Ces deux aspects sont importants pour le développeur qui doit déduire correctement les propriétés importantes. Ils sont aussi importants en cas de discordance entre le système produit et le système souhaité.
- Minimalité : les informations redondantes et les bruits doivent être évités. Les redondances peuvent améliorer la lisibilité mais diminuent la maintenance. Les bruits ne font que diminuer la lisibilité du cahier des charges.
- Précision et non-ambiguïté : « la précision doit empêcher toute interprétation erronée des spécifications, tandis que la non-ambiguïté n'autorise qu'une seule interprétation possible » [Dubru 93]
- Structure : une bonne structure est essentielle à la bonne compréhension de documents pouvant contenir plusieurs centaines de pages. La structure soignée d'un cahier des charges peut être une indication remarquable de sa qualité.

## 2.5 Les problèmes de l'ingénierie des besoins

Cette section donnera quelques indications concernant les problèmes rencontrés lors de l'IB. Le but n'est pas d'être complet mais d'énoncer des difficultés importantes pour la suite.

### 2.5.1 Les différentes cartes du monde

Le client a une idée de ce qu'il aimerait bien avoir. Cette idée est plus ou moins précise, plus ou moins formelle, plus ou moins documentée. Dans tous les cas, elle relève du monde conceptuel du client. Lorsqu'il s'exprime, il le fait dans un contexte d'interprétation qui lui est propre. Il s'exprime en utilisant sa carte du monde (cfr. cours de communication avec N. Habra [Comm97]).

Une des difficultés rencontrées par l'analyste est le fait qu'il ne connaisse pas la carte du monde du client et risque donc d'interpréter certains besoins du client dans un autre contexte. Lorsque le chef d'un département financier d'une entreprise parle du pourcentage de chaque marge de profit associé à chaque produit, l'informaticien sait de quoi il s'agit. Et pourtant, ce pourcentage est-il calculé sur base du prix de revient ou du prix de vente ?

La Figure 4 est une illustration humoristique de ce problème.

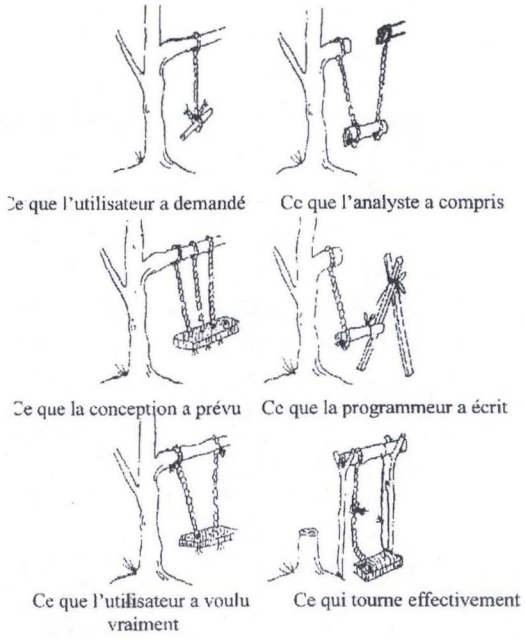



**Figure 4 - illustration du problèmes de compréhension du client [Comm 97]**

Une part importante de l'IB est la compréhension du domaine du client. Il faut que le client et l'analyste se soient mis d'accord sur un même vocabulaire bien défini.

Dans son papier « Four dark corners of requirements engineering », Michael Jackson écrit : « All terminology used in requirements engineering should be grounded in the reality of the environment for which a machine is to be built » [Jackson 96].

### 2.5.2 La détection des problèmes

La détection des incohérences, incomplétudes et ambiguïtés dans un cahier des charges est certainement un des problèmes majeurs de l'analyste. La réussite de cette tâche, s'inscrivant dans l'activité d'analyse selon le modèle de [Dubois 95] ci-dessus, dépend fortement des modèles de représentation utilisés par l'analyste.

Les langages formels apportent une aide appréciable pour détecter des incohérences et ambiguïtés. Le problème est qu'un langage formel est plus limité dans son expressivité par rapport au langage naturel. On essaye alors de rapprocher le plus possible le langage formel aux concepts rencontrés dans le domaine du client.

Cette problématique est fort importante : comment garder une distance minimale entre les concepts exprimés par le client et les mêmes concepts modélisés dans un langage quelconque ? La difficulté réside alors dans la traduction des besoins du monde du client vers le monde de l'analyste, ce qui rejoint la première difficulté expliquée ci-dessus.

### 2.5.3 L'interdépendance entre la technologie, l'organisation et la stratégie

Selon [GTI97], la stratégie d'une entreprise, son organisation et la technologie sont indissociables. En effet, le changement de la stratégie d'une entreprise est supportée par une réorganisation et l'introduction ou la modification de technologies. De même des changements technologiques sans changements organisationnels et sans adaptation de la stratégie n'ont pas beaucoup de chances de succès. La Figure 5 illustre l'interdépendance entre la stratégie, l'organisation et la technologie.

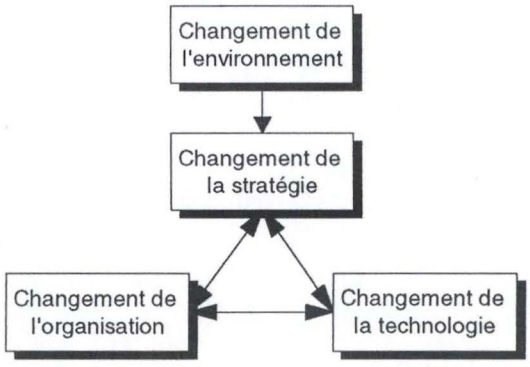



**Figure 5 - l'interdépendance entre stratégie, organisation et technologie**

Un système peut correspondre exactement à ce que le client veut avoir. Le système ne sera pas utilisé si le client ne se réorganise pas un peu en fonction de ce système.

Ainsi, l'analyste doit tenir compte de ces influences lors de l'élaboration du cahier des charges s'il veut que son projet soit un succès.

### 2.5.4 L'existant et le souhaité

On se laisse souvent tenter par une spécification directe du système à réaliser. Cela veut dire qu'on essaye de dire ce qu'un système devrait faire. Une autre manière d'exprimer le 'quoi' est de montrer la différence entre ce qui existe actuellement et ce qu'on aimerait bien qu'il existe après la mise en place de la solution. « It is not necessary or desirable to describe (however abstractly) the machine to be built. Rather, the environment is described in two ways : as it would be without or in spite of the machine, and as we hope because of the machine ». [Jackson 96].

Le problème consiste alors à ne pas se limiter au système devant être réalisé. L'environnement joue un rôle majeur dans l'ingénierie des besoins.

## 2.6 La réduction de ces difficultés

Une manière efficace de réduire les difficultés d'élaboration d'un cahier des charges est de guider le plus possible l'analyste dans sa tâche. Par guider nous voulons dire qu'il faut mettre à la disposition de l'analyste un ensemble d'outils de représentation et de suivis méthodologiques qui lui permettent de réduire les difficultés.

Un modèle de structuration du cahier des charges peut être un tel outil. Cette structure peut, par exemple, contenir les 3 rubriques suivantes : aspects technologiques, aspects organisationnels et aspects stratégiques. Chacune de ces rubriques étant incluse dans un paragraphe « situation actuelle » et un paragraphe « situation souhaitée ». En définissant cette structure de plus en plus détaillée, le risque d'oubli peut être diminué considérablement.

L'élaboration de questionnaires prédéfinis pour des questions plus génériques peut être un autre outil. Disposer d'un ensemble de questions précises peut réduire les difficultés liées à la compréhension du domaine du client.

Le langage de modélisation joue un rôle important dans l'aide de l'analyste. C'est à travers ce langage que l'analyste structurera sa pensée. Il doit pouvoir exprimer des notions proches des concepts employés par le client. L'analyste doit pouvoir exprimer tous les éléments importants à la bonne compréhension des besoins. Cela constitue une limite inférieure pour l'expressivité du langage. D'un autre côté, il est souhaitable que ce langage soit assez formel pour qu'une machine puisse faire des vérifications de cohérence et détecter des ambiguïtés. Cela constitue une limite supérieure de l'expressivité. En plus, ce langage doit être facilement lisible puisqu'il servira aussi à la communication entre l'analyste et le client.

Actuellement, on remarque un intérêt important chez les chercheurs et dans les entreprises pour la recherche de tels outils de soutient.

Le langage ALBERT s'inscrit dans ces recherches. La section suivante rappel en bref les objectifs et les notions élémentaire de ce langage.

## 2.7 Enoncé de l'étude de cas

### 2.7.1 Introduction

Afin de pouvoir rappeler les concepts importants d'ALBERT, nous introduisons l'étude de cas qui va nous servir tout au long de ce mémoire. Il s'agit des besoins fonctionnels d'un système de gestion de carrefours où nous supposons que le client désire régler la circulation grâce à des feux rouges.

Il s'agit de spécifier un système permettant de régler les feux rouges d'un carrefour quelconque dans le sens d'une optimisation des temps d'attente et de la capacité générale du carrefour.

Le système disposera d'informations concrètes sur les dispositions du carrefour qu'il est censé régler. Ces informations expliquent par exemple les dispositions des feux et des senseurs, les temps d'attentes maximales ou encore les temps de réponse estimés des voitures face au passage d'un feu vert au rouge. Le système devra tenir compte de signaux venant d'autres systèmes de gestion de feux rouges afin de pouvoir être coordonné avec ceux-ci. [Benner et al 93]

La modélisation de ce problème est expliquée après l'introduction du langage ALBERT. Ainsi, le lecteur aura l'occasion de se familiariser avec les concepts importants d'ALBERT avant de voir une modélisation dans ce langage. L'annexe 1 reprendra le texte complet de la spécification ALBERT de l'étude de cas.

### 2.7.2 La description du carrefour

Voici le fonctionnement d'un carrefour à feux rouges tel qu'il devrait l'être après l'installation du système.

Les voitures peuvent s'approcher du carrefour en empruntant un tronçon situé dans une certaine direction par rapport au carrefour. Un tronçon est une partie de route débutant au carrefour et partant de celui-ci dans une certaine direction. En général, il y en a 4 mais le système doit être capable de gérer n'importe quelle configuration. Pour chaque tronçon, il existe un certain nombre de voies d'approche (voies qu'empruntent des voitures pour entrer dans le carrefour). Une telle voie d'approche sera appelée une bande. Deux voitures ne peuvent pas se retrouver de front sur une même bande.

A l'intérieur du carrefour, une voiture suit une certaine trajectoire. Une trajectoire relie une bande à un tronçon ne contenant pas cette bande.

Un feu concerne une ou plusieurs trajectoires. Toutes les trajectoires concernées par un même feu doivent être issues de bandes se trouvant sur un même tronçon. Le feu est le dispositif permettant d'ouvrir ou de fermer une trajectoire. Il est caractérisé par 3 couleurs (rouge, orange et vert). La couleur rouge signifie que les trajectoires concernées sont fermées et les couleurs orange et verte signifient que les trajectoires sont ouvertes. Si une trajectoire est fermée, aucune voiture ne peut entrer dans le carrefour pour emprunter cette trajectoire. Si elle est ouverte, les voitures peuvent entrer dans cette trajectoire.

Plusieurs feux peuvent être synchronisés sur une ou plusieurs couleurs. Cela signifie que si un de ces feux a une de ces couleurs, alors les feux synchronisés doivent avoir la même couleur.

Il existe des contraintes d'exclusion sur les feux. Si un feu F1 a une contrainte d'exclusion avec un feu F2, alors F1 et F2 ne peuvent être vert en même temps.

Un carrefour est équipé d'un ou de plusieurs systèmes de détection de présence de voitures (SDPV). Un SDPV peut être installé sur une bande et indique le nombre de voitures

sur cette bande devant le feu. Le nombre maximal de voitures pouvant être détectées dépend du modèle concret de ce système. On peut avoir au maximum un SDPV par bande mais toutes les bandes ne sont pas équipées d'un SDPV.

Une voiture se trouvant à l'intérieur du carrefour peut quitter celui-ci uniquement si aucune autre voiture ne lui bloque le passage. La voiture A se trouvant dans la trajectoire TA bloque la voiture B se trouvant dans la trajectoire TB si TA est bloquante par rapport à TB. On trouve des trajectoires bloquantes au niveau de trajectoires qui se croisent et au niveau de trajectoires issues d'une même bande. Dans le deuxième cas, il faut qu'il y ait un certain nombre de voitures (plus grand ou égal à un) dans la trajectoire bloquante pour que des voitures soient bloquées.

Le système peut recevoir des demandes de l'extérieur pour fermer un tronçon de sortie du carrefour. Cela implique que toutes les trajectoires aboutissantes dans ce tronçon doivent être fermées.

Un carrefour peut être équipé de passages pour piétons. Il en existe alors un ou deux par tronçon. S'il en existe un, celui-ci permet aux piétons de traverser ce tronçon. S'il en existe deux, cela veut dire qu'il existe un îlot au milieu du tronçon qui sépare la partie du tronçon sur lequel les voitures s'approchent du carrefour de la partie du tronçon par lequel des voitures s'éloignent du carrefour. Un passage pour piétons peut être ouvert ou fermé grâce à des feux rouges et dispose de senseurs (boutons-poussoirs) indiquant la présence de piétons. L'ouverture des passages pour piétons est soumise aux mêmes contraintes que les trajectoires : il existe des contraintes d'exclusion interdisant l'ouverture simultanée de certaines trajectoires et de certains passages pour piétons.

Il existe également des contraintes temporelles sur le carrefour :

- Temps entre la commande de fermeture d'une trajectoire et sa libération (temps entre le moment du passage du vert à l'orange et la libération du carrefour par la dernière voiture de la trajectoire en question).
- Temps minimal et maximal pendant lequel les trajectoires doivent être ouvertes et fermées. Ce temps est aussi appelé temps maximal ou minimal d'ouverture ou de fermeture d'une trajectoire.
- Temps orange pour chaque feu.
- Temps maximal d'attente d'une voiture (qui peut être différent du temps maximal de fermeture d'une voie).

A part les contraintes de temps, on trouve aussi des contraintes d'espace :

- Une trajectoire ne peut contenir qu'un nombre maximal de voitures.
- Les voitures ne peuvent entrer dans le carrefour que selon une certaine cadence supposée être constante pour une certaine bande.

- Les voitures ne peuvent quitter le carrefour que selon une certaine cadence supposée être constante pour une certaine trajectoire.

Le système doit optimiser la capacité général du carrefour (le nombre de voitures qui passent par unité de temps) en respectant les contraintes indiquées ci-dessus.

La Figure 6 ci-dessous montre schématiquement un exemple de carrefour pouvant être géré par ce système.

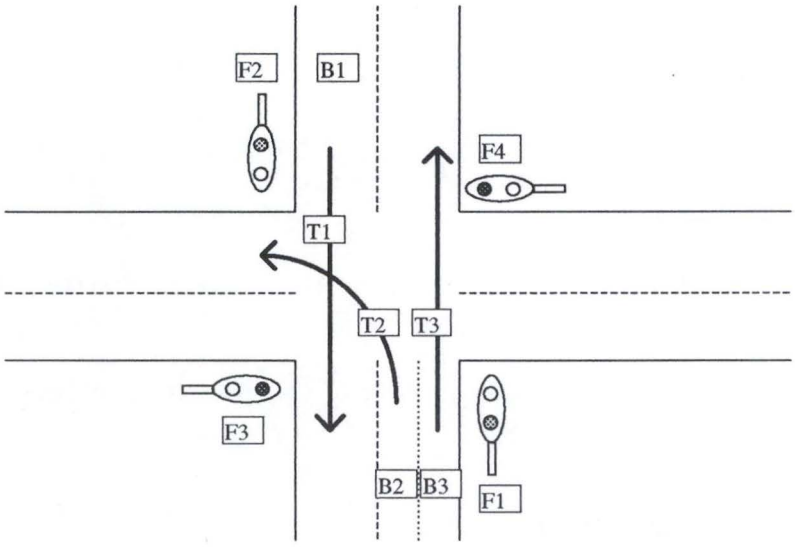



**Figure 6 - exemple de carrefour**

Nous considérerons les trois *bandes* de circulation B1, B2 et B3. Elle donnent respectivement accès aux *trajectoires* T1, T2 et T3. On peut remarquer que les trajectoires T1 et T2 se croisent, on dira que T2 est *ralentie* par T1.

Les feux F1, F2 et F3, F4 forment deux groupes (dans un même groupe, les feux seront verts ou rouges en même temps, ils sont dits *synchronisés*). Les groupes sont dits *exclusifs* dans le sens où les feux qui les composent ne seront jamais verts en même temps.

Quand les feux F1 et F2 passent au vert, les voitures dans les files d'attentes des bandes de circulation peuvent franchir la ligne du feu et se trouvent alors dans le carrefour même. Dans le cas de la trajectoire T3, aucune contrainte n'est imposée aux voitures qui peuvent avancer. Par contre, comme dit précédemment, si des voitures se trouvent dans la trajectoires T1, les voitures venant de T2 seront ralenties.

Enfin, quand le groupe de feux passe au rouge, le deuxième groupe de feux passera au vert après une temporisation de quelques secondes. Et le processus recommence avec les autres bandes et trajectoires (ici perpendiculaires mais non représentées).

## 2.8 Le langage ALBERT

Après une brève introduction (section 2.8.1) et la clarification de la place d'ALBERT dans l'IB (section 2.8.2), nous allons examiner les types d'expressions que l'on trouve chez les clients et montrer où se situe ALBERT par rapport à cela (section 2.8.3). Ensuite, nous introduirons l'étude de cas (section 2.7) qui nous permettra d'expliquer les notions importantes d'ALBERT (section 2.8.4).

### 2.8.1 Introduction

ALBERT est un langage formel d'expression des besoins essayant de combiner la proximité entre les notions du langage et les concepts du monde du client tout en tirant avantage de l'aspect formel.

Le langage ALBERT est particulièrement adapté pour l'analyse d'environnements composites où l'aspect temps réel joue un rôle important.

ALBERT est basé sur la logique temporelle et propose par rapport à elle des extensions permettant surtout de structurer un texte écrit en logique temporelle.

### 2.8.2 La place d'ALBERT dans l'IB

ALBERT est un langage de modélisation. Son but est de couvrir les aspects fonctionnels dans la phase de spécification (cfr. section 2.3 – page 5).

Les aspects fonctionnels se rapportent au fonctionnement et à l'organisation de l'environnement ou de l'entreprise. Ils sont à mettre en opposition avec les besoins non-fonctionnels se rapportant plus aux problèmes stratégiques et à des besoins techniques particuliers. L'imposition de l'utilisation d'un certain interface homme-machine est un exemple de besoin non fonctionnel. La réalisation d'une application facilement maintenable en est un autre.

La phase de spécification comporte les deux tâches de modélisation et d'analyse. ALBERT est un langage de modélisation offrant des outils d'analyse automatisée. La rappelle les tâches de l'ingénierie des besoins et souligne la place d'ALBERT dans ce contexte.

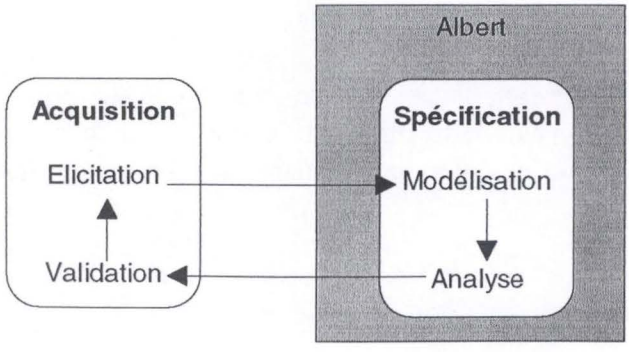



**Figure 7 - la place d'ALBERT dans l'ingénierie des besoins**

### 2.8.3 Les expressions du client

Un des buts d'ALBERT est de proposer des concepts qui sont proches de ceux retrouvés chez les clients. Pour montrer cela, nous essayons de clarifier les manières dont un client s'exprime dans le cadre des besoins fonctionnels.

Le client décrit son environnement grâce à un certain vocabulaire. La description de l'environnement passe surtout par l'énoncé de propriétés liées à cet environnement. Ces propriétés sont exprimées parfois sous forme déclarative et parfois sous forme opérationnelle. En plus des propriétés, on trouve également des morceaux d'exécution ou de scénarios. Ces scénarios décrivent l'évolution de l'environnement pendant un certain laps de temps.

Actuellement, on fait la différence entre un langage permettant d'exprimer le vocabulaire et les propriétés et un langage permettant d'exprimer les scénarios. Les scénarios servent d'une part à trouver le vocabulaire et les propriétés du domaine (élicitation) et d'autre part à la validation de ces propriétés.

Bien qu'ils soient exprimés dans des langages différents, les propriétés et les scénarios sont fortement liés : ils se basent sur un vocabulaire commun et ils expriment le même comportement ! L'idée est donc d'essayer de les intégrer assez fort pour arriver à combiner ces deux modes d'expression. Nous proposons le modèle suivant :

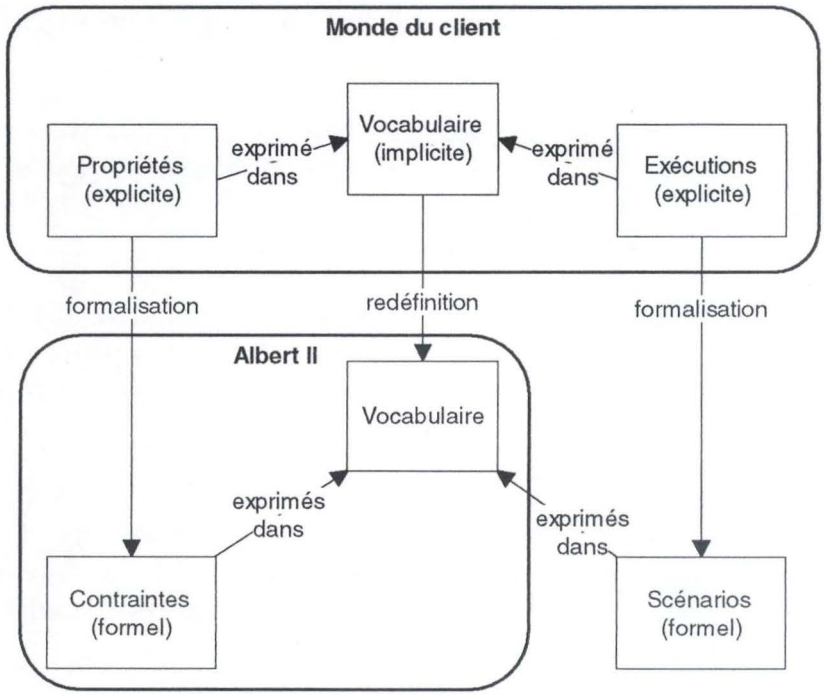



**Figure 8 - modèle des expressions du client**

Le client exprime explicitement les propriétés existantes ou souhaitées de son environnement (domaine) et donne explicitement des exécutions possibles ou indésirables de cet environnement. Il fait cela en utilisant un vocabulaire lié à son environnement. Une redéfinition permet d'arriver à un vocabulaire commun *explicite* servant de base pour l'expression des contraintes et des scénarios. Les contraintes sont les formalisations des

propriétés du domaine exprimées par le client et les scénarios sont les formalisations des exécutions définies par le client.

ALBERT permet de fixer le vocabulaire commun et d'exprimer de manière formelle les contraintes sur l'environnement existant ou souhaité.

Pour souligner la forte liaison entre les propriétés d'un système et les scénarios concernant ce système, nous pouvons remarquer qu'une spécification ALBERT est équivalente à un ensemble de scénarios généralement infini.

### 2.8.4 Les concepts important d'ALBERT

Comme montré sur la Figure 8 page 15, ALBERT permet de définir le vocabulaire lié à l'environnement et les contraintes sur les concepts de l'environnement.

#### A. Le vocabulaire

Une première partie concerne donc la déclaration du vocabulaire. La notion principale est celle d'agent. Un agent peut être vu comme une spécialisation de la notion d'objet dans le sens où un agent est caractérisé par un état et un comportement. Un environnement complexe peut être vu comme un ensemble de parties ayant chacune un comportement spécifique. Le comportement de tout l'environnement est alors obtenu par l'intégration des comportements des différentes parties.

Les agents constituent un excellent moyen de structuration d'un environnement complexe. Dans l'étude de cas, on peut très bien identifier un agent *voiture* décrivant le comportement d'une voiture lorsqu'elle emprunte un carrefour. Le *feu* est un autre agent modélisant le fonctionnement d'un feu rouge dans un carrefour.

ALBERT permet de distinguer un agent isolé ou une classe d'agents. Dans notre cas, *voiture* et *feu* sont tous les deux des classes d'agents. Les agents peuvent être regroupés en des sociétés. Un carrefour peut ainsi être modélisé comme étant une société *carrefour* composée entre autres de plusieurs agents *voiture* et *feu*.

L'état d'un agent est décrit par ses composants d'état. Un composant d'état est une sorte de variable associée à un agent. Ces composants d'état sont typés et on se base sur un ensemble de types prédéfinis, de types déclarés ou construits par l'analyste. L'agent *feu* a, entre autres, un composant d'état nommé *couleur* et modélisant la couleur du feu à chaque moment. Le type *feu* est un type énuméré.

Les valeurs des composants d'états ne peuvent changer que suite à l'occurrence d'une action dont l'agent est responsable. Les actions modélisent le comportement d'un agent. Ils peuvent prendre un certain temps ou peuvent être instantanés. L'agent *feu* a, par exemple, une action nommée passage_*vert* dont l'effet sur le composant d'état est de lui attribuer la valeur *vert*.

Le vocabulaire d'une spécification ALBERT est construit sur base de ces 4 concepts de base : société, agent, composant d'état et action. Les déclarations de ces concepts peuvent être

faites grâce à des représentations textuelles et à des représentations graphiques. Les deux représentations sont équivalentes et le passage entre les deux se fait de manière automatique.

La Figure 9 est la représentation graphique équivalente à la déclaration textuelle suivante de l'agent *voiture* :

**AGENT CARREFOUR.VOITURE**

DECLARATION

STATE COMPONENTS

*Position* instance-of *POSITION_VOIT*

**Intention* instance-of *Carrefour.Trajectoire*

*Ralentie* instance-of *BOOLEAN* derived-from *Intention Position*

ACTIONS

**Approche*

*Entre*

*Sort*

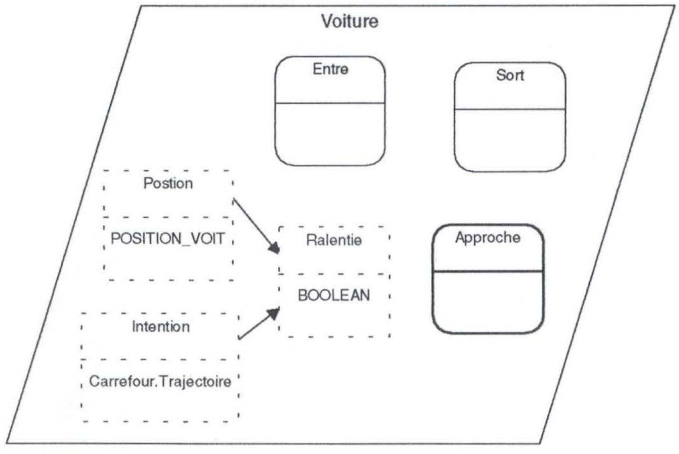



**Figure 9 - représentation graphique de la déclaration de l'agent *voiture***

## B. Les contraintes

Les contraintes sont des phrases écrites dans la logique temporelle des prédicats du 1er ordre et basées sur le vocabulaire déclaré grâce aux moyens expliqués ci-dessus. ALBERT facilite, ici aussi, la structuration de la spécification en proposant différents types de contraintes. Une première structuration est obtenue grâce à la distinction entre contraintes de

base, contraintes déclaratives, contraintes opérationnelles et contraintes de coopération. Ensuite, chaque famille de contraintes en admet des plus fines auxquelles ALBERT associe une certaine syntaxe.

La force de cette structuration réside dans le fait qu'elle permet de classifier et d'exprimer facilement les expressions du client. La traduction vers des contraintes ALBERT des phrases plutôt informelles du client se fait relativement facilement si le vocabulaire a été bien définit.

**Contraintes de base.**

Il existe deux types de contraintes de base.

Premièrement, on peut faire un lien permanent entre différents composants d'états. C'est un mécanisme efficace pour réduire la complexité des expressions des autres contraintes. Dans l'étude de cas, un agent modélise l'état d'une bande du carrefour. Une telle bande est caractérisée par l'ensemble des voitures s'y trouvant. Il est cependant intéressant de savoir qu'elle est la voiture suivante à entrer dans le carrefour. On peut alors déclarer un composant d'état complémentaire nommé voit_*suiv* et dont la valeur est définie par une contrainte de dérivation.

Deuxièmement, ALBERT permet de définir des valuations initiales de composants d'état. Cette contrainte est surtout liée à l'interprétation (cfr. section suivante) que l'on donne à une spécification ALBERT.

**Contraintes déclaratives.**

Au niveau des contraintes déclaratives, nous distinguons les contraintes de comportement d'états, de composition d'action et de durée d'action.

Les contraintes de comportement d'état permettent d'associer un invariant relatif aux composants d'état d'un agent. Par exemple, pour un agent modélisant une trajectoire dans un carrefour, on peut dire que le nombre de voitures dans cette trajectoire est toujours inférieur à une certaine valeur. La validité de l'invariant peut être restreinte à la durée de l'occurrence d'une action. L'invariant peut être défini grâce à des connectivités temporelles. On peut alors exprimer des contraintes du style : « Si le bac d'alimentation est vide, la machine restera à l'arrêt au moins jusqu'à ce que l'ouvrier soit éveillé »[2].

Les contraintes de composition d'action permettent de diviser des actions en des actions plus élémentaires. ALBERT propose des opérateurs de composition d'action couvrant toutes les situations possibles. L'action *passer_carrefour* lié à l'agent *voiture* peut être une composition séquentielle avec intervalle des actions plus élémentaires *approcher*, *entrer* et *sortir*.

---

[2] Exercice lors de la présentation d'ALBERT par Philippe Du Bois dans le cadre du cours MDL matière approfondie.

Les contraintes de durée d'action permettent, comme le nom le dit, de contraindre la durée d'une action. Ainsi, l'entrée d'une voiture dans le carrefour dure au minimum une demi seconde. ALBERT permet d'exprimer ce genre de contrainte facilement.

**Contraintes opérationnelles.**

Les contraintes opérationnelles ressemblent plus à la manière habituelle de travailler pour un informaticien. On distingue les préconditions, les effets d'action et les triggerings.

Les contraintes de préconditions permettent de définir des conditions sous lesquelles des actions peuvent survenir. Ainsi, l'action *entrer* de l'agent *voiture,* modélisant l'entrée de la voiture dans le carrefour, ne peut avoir lieu que si le feu est vert.

Les effets d'actions sont les seules contraintes définissant la manière dont changent les composants d'états. Tout changement d'état est lié à une occurrence d'une action. Ainsi, l'effet de l'action *entrer* du paragraphe ci-dessus est que le composant d'état *position* de l'Agent *voiture* prenne la valeur *dans_carrefour.*

Les contraintes nommées triggerings permettent d'exprimer des conditions sous lesquelles une occurrence d'une certaine action doit avoir lieu. Si nous définissons un temps maximum pendant lequel le feu peut rester rouge, il y aura sûrement une contrainte de type triggering pour déclencher l'action *passage_vert* après ce temps maximal.

**Contraintes de coopération.**

Les contraintes de coopération permettent de créer des liens entre les différents agents. On y trouve des contraintes de mise à disposition d'informations concernant les états et actions d'un agent et d'autre par des contraintes de perception de composants d'état et d'actions d'un autre agent. ALBERT permet d'associer des conditions à ces visibilités et perceptions de composants d'état et d'actions. La Figure 10 montre l'utilité des contraintes de mise à disposition des informations et les contraintes de perception de cette information.

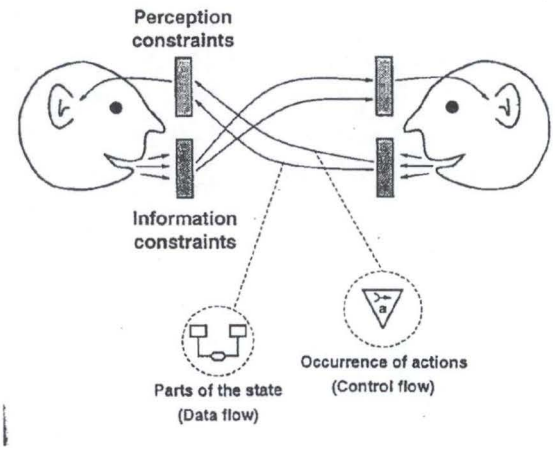



**Figure 10 - illustration des contraintes de coopération [MDLMA97]**

Ainsi, un feu montre ses couleurs à toutes les voitures mais il n'y a que celles qui se trouvent du bon côté qui peuvent les voir.

### C. L'interprétation

Le but d'une spécification est de définir un comportement acceptable de l'environnement. Le comportement acceptable est obtenu grâce à l'intégration des comportements acceptables des différents agents. Un comportement acceptable d'un agent est appelé une vie.

La Figure 11 (inspirée par [MDLMA97]) représente une partie d'une vie d'un agent.

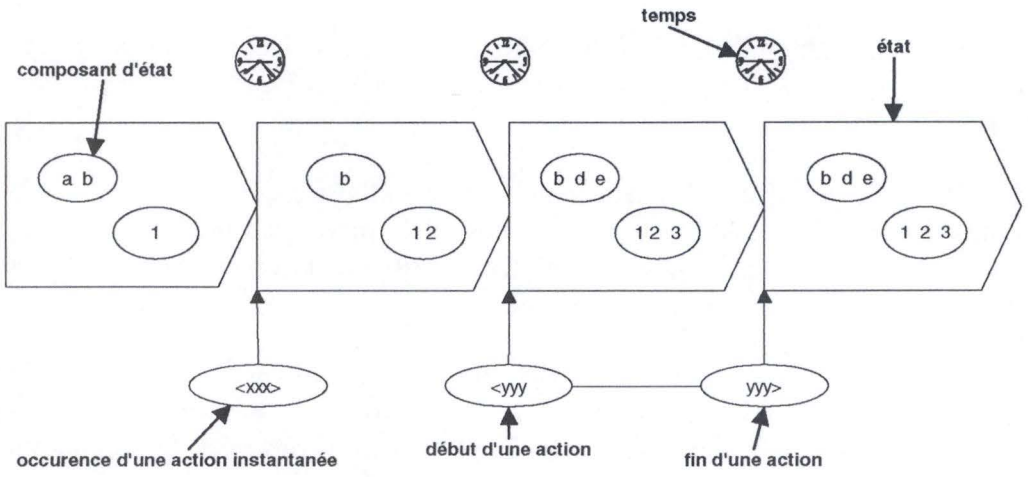



**Figure 11 - représentation d'une partie d'une vie**

Le temps augmente de gauche à droite. L'agent est dans un certain état entre l'occurrence de deux événements. Son état est défini par les valeurs que prennent ses composants d'état. Les seuls événements pouvant changer l'état sont les débuts et les fins d'actions. Chaque occurrence d'un événements a lieu à un certain moment dans le temps. Ce temps est représenté au dessus des occurrences des événements.

La déclaration d'un agent, de ses composants d'état et de ses actions définit une infinité de vies. En effet, l'ensemble des vies de l'agent *feu* avec son composant d'état *couleur* et ses actions *passage_rouge*, *passage_orange* et *passage_vert* est infini puisque les trois actions peuvent avoir lieu à n'importe quel moment. Etant donné une vie, la modification d'un temps de début d'une action engendre une nouvelle vie différente de la première. Comme le nombre de changements possibles est infini, le nombre de vies est infini.

Les contraintes servent alors à restreindre le nombre de vies d'un agent. Chaque écriture d'une contrainte diminue le nombre de comportements acceptables pour l'agent. Si nous reprenons l'exemple de l'agent *feu* ayant un composant d'état *couleur* ainsi que les actions pouvant changer les couleurs, l'introduction d'une contrainte de type *Triggering* forçant le passage au vert après un certain temps élimine toutes les vies ne respectant pas cette contrainte, c'est-à-dire les vies où le composant d'état couleur reste rouge très longtemps.

## 2.8.5 La modélisation de l'étude de cas en ALBERT

Nous avons modélisé le problème du carrefour en utilisant le langage ALBERT. Afin de ne pas trop compliquer l'étude de cas, nous avons posé quelques simplifications et adaptations

par rapport à l'énoncé initial se trouvant à la section 2.7. A l'annexe 1, le lecteur trouvera la modélisation complète du problème du carrefour exprimée dans le langage ALBERT.

La première étape lors de la modélisation est d'identifier les différents agents intervenants dans le problème. Les voitures et les feux sont des candidats prometteurs tout comme l'agent système. Cependant, notre choix est de ne pas tellement décrire comment fonctionnerait le *système* mais plutôt comment nous aimerions que fonctionne le *carrefour* après l'installation de ce système. Cela nous pousse à ne pas prendre le système comme un agent de la modélisation mais de distribuer 'l'intelligence' de celui-ci sur les feux. Ainsi, nous nous limitons à la description du fonctionnement de l'environnement du système.

Les deux agents 'feu' et 'voiture' sont effectivement les éléments de l'environnement réalisant les actions. Il ne faut cependant pas oublier les carrefours environnants pouvants avoir des influences sur le carrefour actuel. Nous introduisons donc un autre agent de nom 'autre_carrefour'.

L'énoncé du problème prévoit l'utilisation du système dans n'importe quelle configuration de carrefour. Cela nous pose des problèmes puisqu'il faut quelque part définir une structure permettant de représenter n'importe quel carrefour. Notre première idée est de définir un agent 'carrefour' ayant les composants d'état adéquats pour représenter cette structure. Cependant, cela pose plusieurs problèmes. Le plus important est que la structure des composants d'état devient très compliquée et très technique. On s'écarte des concepts exprimés par le client pour arriver à des concepts liés à une technique de modélisation.

Notre solution est de définir deux agents 'passifs' (sans actions, uniquement des composants d'état) 'bande' et 'trajectoire' pouvants représenter la configuration d'un carrefour concret. Les composants d'état deviennent plus petits et correspondent mieux à des concepts exprimés par le client.

Nous identifions donc 5 agents, tous pouvant avoir plusieurs instances : voiture, feu, bande, trajectoire et autre_carrefour.

Notre deuxième étape consiste à représenter les différents liens entre ces agents. Cela est indispensable pour décrire la configuration du carrefour.

Pour commencer, nous décrivons le comportement d'une voiture par rapport à sa position au carrefour. Les positions intéressantes de la voiture sont 'loin avant', 'devant le feu', 'dans le carrefour' et 'loin après le carrefour'. Une voiture passe d'une position à une autre respectivement en s'approchant, en entrant et en sortant du carrefour. Sa position initiale est 'loin avant'. Ces considérations donnent lieu, au niveau de l'agent voiture, à un composant d'état 'position' et à des actions 'approche', 'entre' et 'sort'. Afin de pouvoir modéliser qu'il n'y ait pas trop de voitures qui entrent et sortent en même du carrefour, les actions 'entre' et 'sort' prennent un certain temps.

Ensuite, nous devons assurer la contrainte que les voitures ne se dépassent pas dans les bandes et dans les trajectoires. En plus, nous devons pouvoir repérer facilement les voitures dans une trajectoire, la prochaine voiture pouvant entrer dans le carrefour sur une certaine bande et la prochaine voiture pouvant sortir du carrefour sur une certaine trajectoire. Nous modélisons cela par l'introduction de files dans les bandes et les trajectoires, ce qui correspond

à la réalité d'ailleurs. Une file est représentée par un composant d'état de type séquence. Nous introduisons aussi un composant d'état dérivé de cette file représentant la prochaine voiture à pouvoir entrer ou sortir du carrefour sur une certaine bande ou trajectoire.

Des contraintes de visibilité des actions de la voiture assurent que les agents 'bande' et 'trajectoire' peuvent mettre à jour la file d'attente et que l'intégrité soit ainsi conservée.

Le comportement des feux est notre préoccupation suivante. Les feux ont une certaine couleur (rouge, orange ou vert) qui sera représentée par un composant d'état de même nom. Des actions instantanées assurent le passage d'une couleur à une autre. Dans l'énoncé, on parle explicitement de temps minimal et maximal d'affichage de chaque couleur. Afin de garder une certaine flexibilité, nous choisissons de représenter ces temps par des variables devant être instancié en même temps que la spécification. La raison principale est que ces temps minimaux et maximaux dépendent du carrefour concret. Nous trouvons donc 6 composants d'état constants pour l'agent feu qui représentent ces temps minimaux et maximaux. Des contraintes de 'précondition' et de 'triggering' assurent que ces temps soient respectés.

Ensuite, nous restreignons le comportement des voitures par rapport aux feux. Les voitures peuvent voir le feu qui les concerne et s'arrêtent lorsqu'il est rouge. Le premier problème à résoudre est de savoir quelle voiture est concernée par quel feu. Pour résoudre ce problème, nous introduisons l'intention d'une voiture qui correspond à la trajectoire que cette voiture empruntera. Chaque trajectoire est régie par un seul feu. Nous ajoutons donc un composant d'état au niveau des trajectoires indiquant quel feu régit cette trajectoire. Grâce à ces deux éléments supplémentaires ('intention' au niveau voiture et 'feu régi' au niveau trajectoire), nous sommes capables d'écrire les contraintes assurant la visibilité du bon feu envers les voitures.

Il nous suffit donc d'écrire une précondition pour l'action 'entrer' dans le carrefour des voitures les empêchant de rentrer lorsque le feu est rouge. Nous profitons de l'occasion pour écrire la contrainte qu'uniquement la voiture suivante peut entrer dans le carrefour. Cela empêche le doublement des voitures dans les bandes. Comme c'est la même chose au niveau des trajectoires, nous ajoutons la contrainte adéquate pour l'action 'sortir' du carrefour.

Nous rencontrons maintenant une autre grande difficulté de modélisation : comment exprimer les contraintes de synchronisation entre les feux ?. Premièrement, nous devons savoir quels feux sont synchronisés entre eux. Comme cela dépend de la configuration exacte du carrefour, nous ajoutons au niveau de l'agent feu un composant d'état reprenant la liste des feux synchronisés avec lui et sur quelle couleur. Il faut aussi que les feux voient les couleurs des autres feux pour qu'ils puissent se synchroniser. Une contrainte de type 'state behavior' assure alors que les couleurs des feux synchronisés soient toujours les mêmes.

Les contraintes d'exclusion sont du même type. Il faut d'abord savoir quel feu à une contrainte d'exclusion avec quel autre feu. Cela provoque la création du composant d'état 'exclusion' reprenant tous les feux ayant une contrainte d'exclusion avec l'actuel. Une contrainte de type 'state behavior' assure alors que deux feux ayant une contrainte d'exclusion ne soient pas verts en même temps.

L'étape suivante consiste à tenir compte des systèmes de détection de présence de voitures (SDPV). Nous décidons de les modéliser uniquement au niveau des bandes devant les

feux. Une première chose est de spécifier la capacité des SDPV pour chaque bande. Comme cela dépend du carrefour effectif, nous introduisons un composant d'état constant représentant cette valeur. Pour une bande n'ayant pas de SDPV, ce nombre vaut zéro. Ensuite, le composant d'état représentant le nombre de voitures effectivement détectées se dérive directement du nombre maximal détectable et du nombre de voitures se trouvant dans la file associée à la bande.

Le problème est alors de modéliser l'influence du nombre de voitures détectées dans une bande sur le fonctionnement des feux. Nous avons décidé d'introduire le concept de priorité liée à une bande et liée à un feu. La priorité d'une bande est simplement la proportion de voitures détectées par rapport aux voitures détectables. Ainsi, plus il y a de voitures, plus haut est la priorité de la bande. S'il n'y a pas de SDPV, la priorité est toujours nulle. Cela est un choix de design qui normalement ne doit pas être pris à ce niveau-ci mais nous n'avons pas trouvé d'autres solutions.

La priorité d'un feu est alors le maximum des priorités des bandes qu'il concerne. Nous aurions très bien pu choisir la moyenne des priorités des bandes mais l'énoncé ne donne pas d'indication sur ce choix et nous prenons le plus facile.

Une contrainte de type 'triggering' assure qu'un feu ayant une priorité plus basse que tous les autres feux ayant une contrainte d'exclusion avec lui, passe rapidement au rouge.

Le dernier élément à modéliser est le fait qu'un feu doit être rouge lorsqu'il règle une trajectoire qui donne sur un autre carrefour bloqué. Pour cela, nous introduisons une contrainte de visibilité permettant au feu de voir quand ce carrefour est bloqué. Une contrainte de type 'state behavior' assure alors la couleur rouge du feu pendant le blocage du carrefour concerné.

Durant toute la modélisation, nous avons simplifié certains éléments de l'énoncé informel. Nous allons maintenant énumérer ces simplifications.

- La notion de tronçon n'est pas modélisée.
- Pas de respect de la contrainte « toutes les trajectoires concernées par un même feu doivent être issues de bandes se trouvant sur un même tronçon ».
- Prise de choix de design par rapport au traitement de l'information reçue par les SDPV.
- Les trajectoires bloquantes ont été remplacées par des trajectoires ralentissantes pour éviter des problèmes de blocage mutuel.
- Les voitures venant d'une bande pour aller tout droit ne sont pas ralenties par des voitures venant de la même bande mais voulant aller vers la gauche, comme c'est prévu dans l'énoncé informel.
- Les passages pour piétons ne sont pas modélisés.
- Le temps entre la fermeture d'une trajectoire et la libération de celle-ci n'est pas spécifié.
- Le temps maximal d'attente d'une voiture n'est pas spécifié.
- Le but du troughput maximal n'est pas spécifié.

# 3. Les scénarios

## 3.1 Introduction

Le but de ce section est de préciser le concept de scénario sans faire la correspondance avec le langage ALBERT.

Il existe des notions fortement liées à celle de scénario. Les « use case », les prototypes, l'animation et la simulation sont de telles notions qui seront définies et discutées à la section 3.2.

L'utilité des scénarios ne se limite pas uniquement à l'élicitation et à la validation des besoins. La section 3.3 donnera d'autres utilités couvrant presque tout le cycle de vie du développement de logiciels.

Afin de pouvoir classifier mais aussi pour pouvoir déterminer des caractéristiques importantes relatives à la notation et au contenu des scénarios, les sections 3.4 et 3.5 proposent une analyse de la représentation et du contenu des scénarios.

Suite à cela, nous identifieront quelques problèmes souvent rencontrés lors de l'élaboration de scénarios (section 3.6) et nous terminerons par plusieurs exemples tirés de la littérature ou inventés dans le cadre de l'étude de cas (section 3.7).

## 3.2 Définitions

En fonction du contexte et du centre d'intérêt des personnes travaillant dans le domaine des scénarios, la compréhension du terme change considérablement. Des nouveaux termes sont inventés pour mieux décrire les formes particulières de scénario (ex. « use case » [Jacobson 92]). Certaines techniques semblent utiliser des scénarios sans pour autant employer le mot scénario. Le but de cette section est de définir ce que nous entendons par scénario et de comparer cette définition aux notions liées.

### 3.2.1 Scénario

Le Micro Robert [Micro Robert 78] définit le scénario dans le cadre d'un film comme suit : « Description de l'action (d'un film), comprenant généralement des indications techniques et des dialogues ».

Cette définition reste valable dans le cadre de l'ingénierie des besoins si nous considérons le fonctionnement d'un système dans son environnement comme une sorte de film se déroulant dans le temps.

Il est intéressant de souligner les deux mots clés *action* et *dialogue*. Un scénario est constitué d'une suite d'actions et porte une attention particulière aux dialogues entre les acteurs intervenants.

Si nous transposons cette définition dans le contexte d'une spécification en ALBERT, on peut dire qu'un scénario ALBERT est une suite d'actions réalisées par des agents coopérants grâce à des échanges de messages. Plus précisément, un scénario décrit un ensemble de vies ALBERT faisant intervenir plusieurs agents.

### 3.2.2 Use case

Le terme fort à la mode pour l'instant est « use case ». Il a été introduit par Ivar Jacobson lors de la conférence OOPSLA 1987 [Berard]. Bien que le principe se cachant derrière les « use case » existait déjà avant, la notion décrite par Ivar Jacobson connaît un succès important seulement maintenant. Voici comment il définit les « use case » :

"A use case is a specific way of using the system by performing some part of the functionality. Each use case constitutes a complete course of events initiated by an actor and it specifies the interaction that takes place between an actor and the system. A use case is thus a special sequence of related transactions performed by an actor and the system in dialogue." [Jacobson 92]

Dans [Jacobson 94a] on trouve une caractérisation intéressante des « use case » :

"A good use case when instantiated is a sequence of transactions performed by a system, which yields a measurable result of values for a particular actor."

Cette caractéristique se rapporte plus à la sémantique des « use case ». Les deux groupes de mots soulignés donnent respectivement une sorte de limite inférieure et supérieure à la portée d'un « use case ». En effet, un « use case » apporte au minimum une valeur mesurable à au maximum un acteur particulier. L'instance d'un « use case » est appelé un scénario.

Si nous comparons la notion de « use case » à notre compréhension de la notion de scénario, nous pouvons identifier au moins la différence suivante : les scénarios sont plus générales vu qu'ils ne décrivent pas l'interaction entre l'utilisateur et le système mais l'interaction et le comportement de n'importe quels agents. Le système peut être constitué de plusieurs agents et l'environnement du système peut être fait d'autres choses que d'utilisateurs.

Concrètement, dans le cadre de notre exemple du feu rouge, un scénario décrit l'interaction et le comportement d'une ou de plusieurs voitures, d'un ou de plusieurs feux, d'un ou de plusieurs carrefours environnants, etc. Un « use case » décrit uniquement le comportement d'une voiture et son interaction avec l'interface du système, c'est-à-dire les feux. Les voitures et les carrefours environnants font partie de l'environnement et les feux font partie de l'interface entre l'environnement et le système.

La Figure 12 illustre la distinction entre le système, l'interface et l'environnement dans le cadre du problème du carrefour.

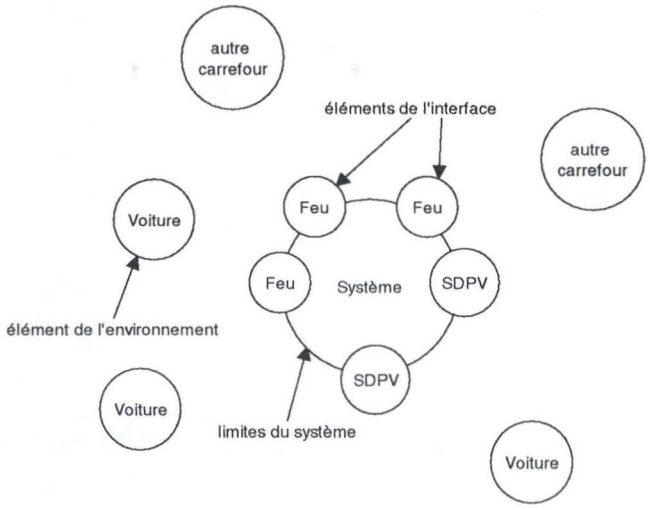



**Figure 12 - distinction entre système, interface et environnement**

On trouve une autre différence importante entre la notion de scénario et la notion de « use case ». D'après Ivar Jacobson, le but de l'ensemble des « use case » est de former la spécification. Comme expliqué plus loin, nous voyons le but des scénarios plus dans l'élicitation et la validation d'une spécification mais pas pour former la specification elle-même.

### 3.2.3 Prototype

Le Micro Robert [Micro Robert 78] définit un prototype comme suit : « Premier exemplaire d'un modèle (de mécanisme, de véhicule) construit avant la fabrication en série. »

Dans le cadre de l'ingénierie des besoins, un prototype est un système partiel permettant de tester certaines caractéristiques attendues du système complet avant que celui-ci ne soit implémenté. Le principe est de se baser sur la spécification d'un système pour l'exécuter sans devoir passer par des phases de design et d'implémentation.

Un prototype est plus ou moins proche du système souhaité. Deux facteurs souvent opposés déterminent cette proximité : le temps dont on dispose pour réaliser le prototype et les caractéristiques souhaitées du prototype.

Grâce à cette technique, l'utilisateur a la possibilité de valider les parties des spécifications qui semblent être critiques pour la bonne réussite du projet. Pour ce faire, il parcourt plusieurs scénarios et observe la façon dont le prototype se comporte. C'est dans ce sens que nous pouvons dire que la technique du prototype est liée aux scénarios.

On pourrait dire que les scénarios font partie de l'entrée et de la sortie d'une session de test utilisant la technique des prototypes.

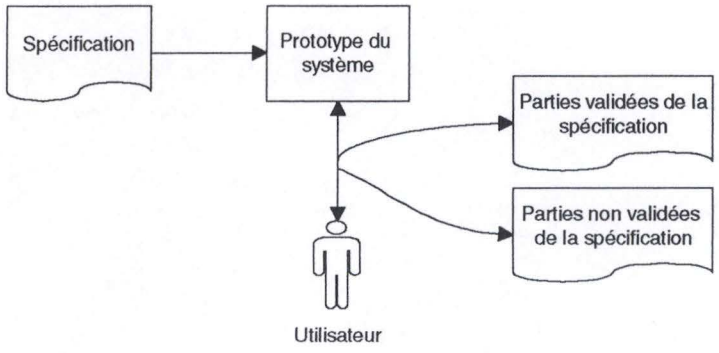



**Figure 13 - Utilisation de prototypes**

La Figure 13 dessine schématiquement le fonctionnement de la technique des prototypes. Une spécification est transformée en un système simplifié. Celui-ci est exécuté par un utilisateur qui teste ses scénarios. En fonction du comportement du prototype, l'analyste peut déterminer des problèmes de spécification.

### 3.2.4 Animation

La technique des prototypes est très efficace mais elle demande l'existence d'un logiciel qui puisse exécuter une spécification. Cependant, les spécifications ne sont pas toujours exécutables [Hayes 90]. Cela est surtout dû au fait qu'une spécification comporte un certain nombre d'indéterminations. Admettre de l'indétermination n'est pas un désavantage. Au contraire, il permet d'écrire des spécifications fort concises laissant des choix de design à la phase de design du système. Si l'indétermination est intéressante au niveau des spécifications, elle empêche la création d'un prototype. L'animation permet de remédier à ce problème de l'indétermination.

L'animation est une technique qui permet d'avancer pas à pas dans un scénario tout en respectant la spécification du système. Les scénarios ne doivent pas nécessairement être connus à l'avance, ils peuvent être construits en cours d'animation. Lorsque des indéterminations se présentent, l'utilisateur peut les enlever immédiatement.

L'animation sert à valider une spécification en proposant à l'utilisateur de « l'expérimenter ». Le résultat d'une animation est donc un ensemble de critiques relatives à la spécification. Le schéma Figure 14 - Fonctionnement d'une animation (Figure 14) montre de manière simplifiée le fonctionnement d'une animation.

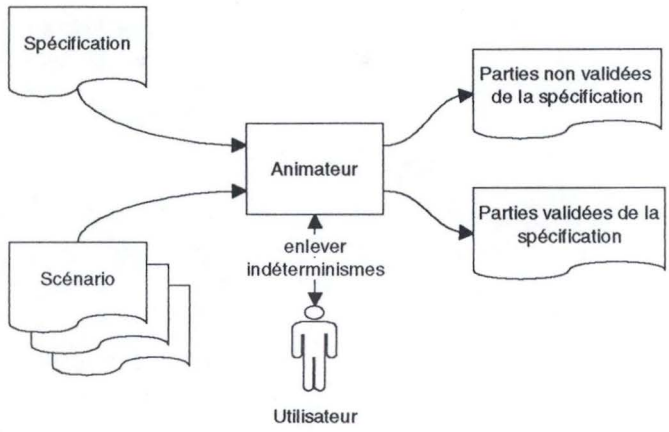



**Figure 14 - Fonctionnement d'une animation**

Un programme appelé animateur prend en entré une spécification et éventuellement la description d'un ou de plusieurs scénarios. Il exécute les scénarios en respectant la spécification et demande régulièrement à l'utilisateur d'enlever des indéterminations.

Un scénario peut aussi être le résultat d'une animation. En effet, après une session d'animation, l'utilisateur aura parcouru un ou plusieurs scénarios et aura enlevé toutes les indéterminations. La spécification aura alors été validée par rapport à ces scénarios.

### 3.2.5 Simulation

On peut définir la simulation comme l'exécution automatique d'un modèle aussi proche que possible de la réalité.

A priori, le mot simulation pourrait faire penser à une animation ou à un prototypage sans intervention de l'utilisateur. Le comportement de l'utilisateur serait alors remplacé par une exploration de tous les comportements possibles et/ou par des statistiques relatives au comportement normal ou anormal de l'utilisateur.

En réalité, la simulation est plutôt utilisée pour déterminer certains paramètres lors de la prise de décisions importantes relatives au comportement du système. Dans le contexte du système de feux, une simulation pourrait par exemple aider à déterminer les paramètres de temps vert et temps rouge pour chaque feu.

## *3.3 L'utilité des scénarios*

### 3.3.1 Introduction

Les scénarios peuvent être employés à plusieurs moments dans le cycle de vie du développement d'un logiciel ainsi qu'à d'autres occasions. Dans une première section, nous rappellerons les différentes étapes rencontrées lors du développement d'un logiciel (section 3.3.2), ensuite nous montrerons comment la technique des scénarios peut être utilisée lors de

ces différentes étapes (section 3.3.3). Finalement, nous donnerons quelques autres utilités (section 3.3.4).

### 3.3.2 Le cycle de vie d'un logiciel

A la section 2.3, nous avons abordé le processus menant à l'expression des besoins. Il est essentiellement basé sur 4 phases différentes parcourues de manière itérative : élicitation, modélisation, analyse et validation.

Des modélisations semblables existent pour toutes les activités intervenants dans le cycle de vie d'un logiciel. Comme déjà énoncé dans l'introduction (section 1.1), on identifie généralement les phases suivantes : l'analyse, le design, l'implémentation, les tests et le déploiement.

Dans ce contexte, la phase d'analyse correspond à ce que nous appelons l'ingénierie des besoins. Elle se termine lorsque le cahier des charges est accepté par le fournisseur et le client. Par ce fait, elle peut être facilement distinguée dans le temps par rapport aux autres phases. En effet, en principe, le design ne commence que lorsque le cahier des charges est fixé correctement.

Le design consiste en l'élaboration d'une architecture du système à réaliser. Il s'agit de faire une découpe des fonctionnalités en parties plus petites et facilement gérables tout en respectant le cahier des charges et en assurant les qualités d'un 'bon' logiciel. Rappelons qu'un 'bon' logiciel est facilement modifiable, sans erreur, facilement utilisable, efficace, rapide et peu gourmand en ressources.

Grâce à un design bien fait, la phase d'implémentation peut être automatisée assez fortement. L'implémentation consiste en l'écriture du code exécutable par un ordinateur dans le cas d'un logiciel. Lorsqu'il s'agit de mettre en place de nouvelles procédures de travail à l'intérieur d'une entreprise, la phase d'implémentation peut être l'écriture des manuels de procédure.

La phase de test peut être vue comme une remontée des phases énoncées ci-avant. Elle commence par la falsification des différentes parties implémentées. Ensuite, il s'agit de voir si les différentes parties de l'implémentation fonctionnent bien ensemble. Cela constitue un sorte de falsification du design. Finalement, on essaye de voir si le produit implémenté correspond bien au cahier des charges. Vu cette remontée des différentes phases préalables, on trouve dans certains ouvrages le modèle en V représenté sur la Figure 15. Il illustre la correspondance entre les phases d'analyse, de design et d'implémentation et les différents types de tests.

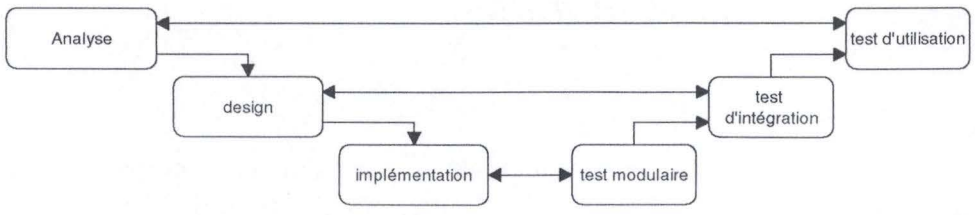



**Figure 15 - modèle en V**

Lorsque le système a été testé et le cas échéant corrigé, on passe au déploiement de celui-ci. Le système sera alors placé dans son environnement réel et fonctionnera idéalement correctement à partir de ce moment.

### 3.3.3 Support au cycle de vie d'un logiciel

L'exploitation de scénarios est une technique intéressante à beaucoup d'instants dans le cycle de vie d'une logiciel. Elle sert aussi bien lors de la recherche de solutions possibles qu'à la validation de celles-ci.

Comme nous l'avons déjà énoncé plusieurs fois, les scénarios sont très intéressants lors de l'acquisition des besoins (acquisition = élicitation + validation). Ils permettent à l'analyste de découvrir des propriétés intéressantes du système et de voir comment le client aimerais bien l'utiliser.

Les scénarios permettent aussi de découvrir le vocabulaire important du domaine d'application (exemples de domaines d'application : le domaine financier, le CIM[3], l'alimentation, etc.).

Au niveau de la validation, les scénarios permettent de voir si une spécification correspond au désirs du client. Si celui-ci décrit un scénario acceptable pour lui mais non accepté par la spécification, alors l'analyste doit modifier sa spécification.

Au niveau de la phase de design, les scénarios permettent l'évaluation de plusieurs possibilités lors d'un choix important à prendre. Comme indiqué à la section 3.2.5, des scénarios peuvent être à la base d'une stratégie de simulation. Cette simulation permettra alors de voir quelle est la possibilité la plus intéressante par rapport à ces scénarios.

Les scénarios redeviennent très importants lors de la phase de test. Surtout lorsqu'il s'agit de la validation du système par rapport au cahier des charges, un ensemble de scénarios est d'une très grande utilité. Certains auteurs [Jacobson 92] proposent d'utiliser pour les tests les scénarios employés lors de l'ingénierie des besoins. Les scénarios ayant servis à la validation de la spécification seront alors réutilisés lors de la validation du système par rapport aux spécifications.

### 3.3.4 Autres utilités

Les scénarios semblent être très intéressants lorsqu'il s'agit de décrire un déroulement dans le temps faisant intervenir plusieurs acteurs. Ils peuvent donc aussi servir à la définition de procédures comme on les trouve dans les manuels d'utilisateurs livrés avec des logiciels. En fonction des tâches que veut accomplir l'utilisateur, il choisit le scénario adéquat, s'identifie avec un acteur intervenant dans ce scénario et exécute la procédure décrite.

On peut dire de même pour ce qui est de la description des procédures par les différentes méthodologies de développement de logiciels. Le modèle « waterfall » ou celui de la « spirale »

---

[3] CIM = Computer Integrated Manufacturing

(cfr. [Jacobson 92] pour une description de ces deux modèles) ne sont rien d'autre que de scénarios abstraits. Ils sont abstrait dans le sens où ils sont instanciés à chaque projet concret utilisant cette méthodologie.

Le modèle de la spirale consiste à voir le processus de développement de logiciels de manière itérative. L'analyste réalise d'abord une partie du cahier des charges (en général les fonctionnalités essentielles) et le valide avec le client. Ensuite, les autres phases sont exécutées pour finalement donner lieu à un premier logiciel réalisant les fonctionnalités de base. Une fois accepté, l'analyste intègre une nouvelle fonctionnalité au cahier des charges. Elle passe par les différentes phases qui l'intègrent au produit déjà existant. Ainsi de manière itérative et incrémentale, le logiciel complet se construit. La Figure 16 montre ce processus en spirale.

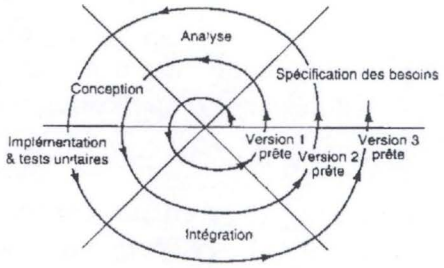



**Figure 16 - le modèle de la spirale [Jacobson 92]**

Dans ce contexte, les scénarios peuvent à nouveau être d'une grande utilité. Ils peuvent en effet constituer une unité d'itération. Chaque nouvelle itération correspond à l'ajout d'une fonctionnalité correspondant à un ou plusieurs scénarios.

## 3.4 La représentation

### 3.4.1 Introduction

Dans leur rapport D1-I1.1 [CREWSD1-I1.1], le projet Esprit CREWS (Cooperative Requirements Engineering With Scenarios) propose une grille de classification de scénarios. Chaque scénario est caractérisé selon 4 dimensions : sa forme, son contenu, son but et son cycle de vie. Chaque dimension correspond respectivement aux questions suivantes :

- Sous quelle forme un scénario est-il présenté ?
- Quels informations sont exprimées dans le scénario ?
- Pourquoi utiliser un scénario ?
- Comment manipuler un scénario ?

La Figure 17 illustre le cadre de classification des scénarios définit dans [CREWSD1-I1.1].

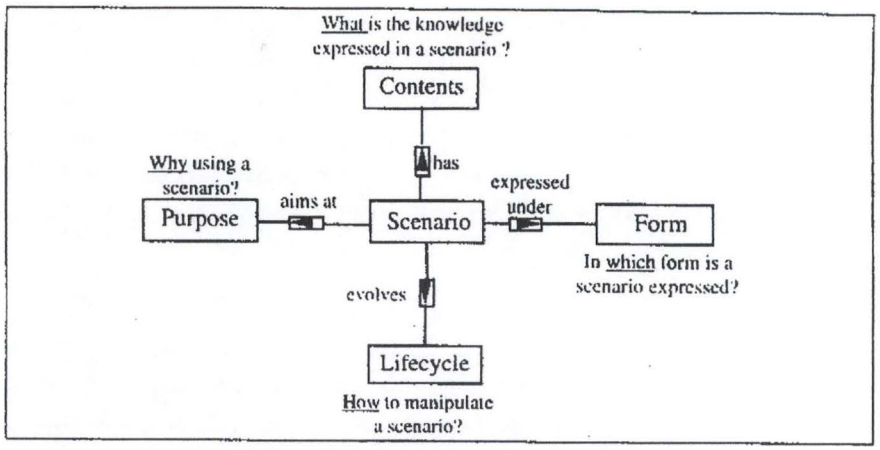



**Figure 1 : The four views on scenarios**

**Figure 17 - cadre pour la classification des scénarios [CREWS D1-I1.1]**

Le rapport applique cette grille d'analyse à différents type de scénarios existants, comme par exemple, les « use case » introduits à la section 3.2.2.

Dans ce paragraphe, nous allons nous intéresser un peu plus aux formes des scénarios en apportant une certaine extension par rapport aux critères proposés par l'équipe CREWS. La section suivante (3.5) détaillera un peu plus la dimension 'contenu'.

### 3.4.2 Les dimensions

L'équipe CREWS distingue 2 facettes à l'intérieur de la dimension 'forme'. Premièrement la description et deuxièmement le présentation.

La description classifie les scénarios selon le médium sur lequel est exprimé le scénario et le style de notation utilisé. Le médium peut être du texte, des graphiques, des images, de la vidéo, un prototype et toute combinaison de ces éléments. La notation peut être formelle, semi-formelle et informelle.

La présentation classifie les scénarios selon qu'ils peuvent être animés ou non et selon leur interactivité. L'interactivité peut être non existante, du type hypertexte ou du type avancé. Pour plus d'informations sur ces éléments de représentation, le lecteur pourra consulter [CREWSD1-I1.1] à la page 9.

Nous allons garder l'idée du médium et de la notation formelle ou informelle ainsi que le niveau d'interactivité lors de la représentation des scénarios. A ces trois dimensions viendront s'ajouter le niveau de détail et la structuration entre différents scénarios.

Ces 5 dimensions relativement indépendantes détermineront la manière dont les scénarios peuvent être représentés. Dans les sections suivantes, nous les détaillerons un peu plus en faisant référence au « use case » comme exemples.

## A. Le médium

La représentation initiale des « use case » est un mélange entre des dessins élémentaires et du texte. [Jacobson 92] propose des icônes pour identifier les acteurs, les « use case » et les relations entre acteurs et « use case ». La description textuelle se rapporte essentiellement à la représentation des suites d'actions et d'interactions entre les acteurs (cfr. encart ci-dessous) et le système alors que la partie graphique se rapporte plus à la structuration entre les « use case » (Figure 19). [UML] et [Andersson 95] proposent également des représentations graphiques pour la suite des actions (cfr. Figure 20).

> Lorsque le client dépose un article consigné, ce dernier est mesuré afin de déterminer de quel type de bouteille, de boîte ou de cageot il s'agit. Si l'article est accepté, le total du client est incrémenté, de même que le total journalier concernant ce type d'élément particulier. Si l'article n'est pas accepté, la lumière „NON VALIDE" s'allume sur le panneau.
>
> Lorsque le client appuie sur le bouton de reçu, l'imprimante commence à imprimer la date. On calcule le total du client et on imprime les informations suivantes sur le reçu, pour chaque type d'article:
>
> Nom
>
> Nombre de dépôts effectués
>
> Valeur de dépôt pour le type d'article
>
> Total pour ce type
>
> Et enfin la somme que le client doit recevoir est imprimé sur le reçu

**Figure 18 - exemple de représentation textuelle de la suite des actions d'un scénario [Jacobson 92]**

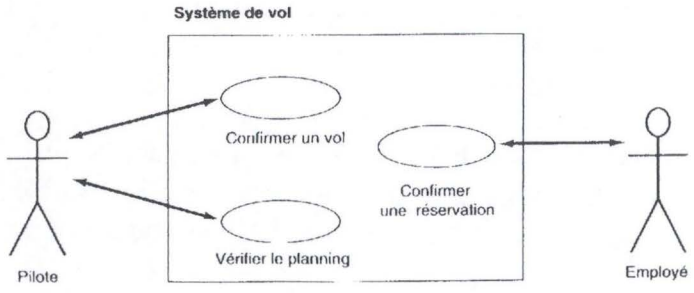



**Figure 19 - exemple de représentation graphique de la structuration de "use case" [Jacobson 92]**

La Figure 19 montre un exemple de représentation graphique de la structuration de « use cases ». Les formes ovales représentent des « use case » et les hommes dessinés en trait représentent des acteurs.

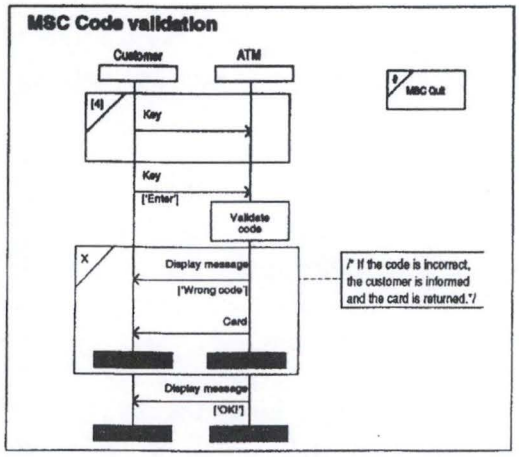



**Figure 20 - exemple de représentation graphique de la suite des actions [Andersson 95]**

La Figure 20 donne un exemple de représentation graphique de la suite des actions d'un scénario basé sur le formalisme des MSC (message sequence charts). Les deux acteurs sont représentés en colonnes et l'exécution dans le temps de haut en bas. Nous allons décrire en détail le formalisme des MSC à la section 4.2.1.

A part ces deux possibilités, les scénarios peuvent également être représentés en employant des images. Cela peut être particulièrement intéressant lors de la description d'une session d'utilisation d'un logiciel. En effet, l'image de l'interface homme-machine facilite la lecture d'un tel scénario. La Figure 21 montre un exemple de dessin d'un interface lié à un scénario.

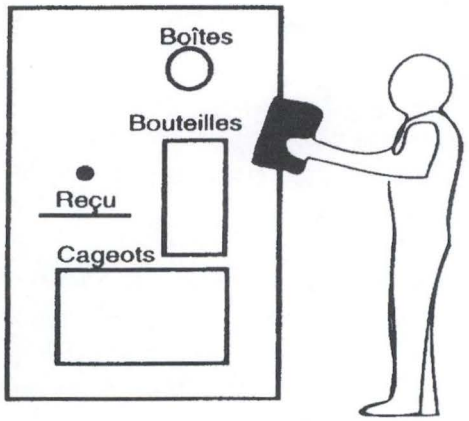



**Figure 21 - exemple de dessin d'interface [Jacobson 92]**

Pour l'acquisition des scénarios, un support vidéo est d'une grande efficacité. En effet, filmer les utilisateurs en pleine action permettra d'éliminer des négligences rencontrées lorsque le client doit raconter ce qu'il fait et comment il le fait. Des étapes qui semblent être évidentes pour le client ne le sont peut-être pas pour l'analyste.

Comme médium, nous distinguons donc texte, graphique, image et vidéo.

### B. Le niveau de formalité

Une représentation est dite formelle lorsqu'elle peut être exploitée facilement par une machine. Par exploitation nous entendons non seulement la manipulation mais aussi des opérations du type vérification de contraintes, validation par rapport à une spécification et génération assez immédiate de prototypes.

Actuellement la plupart des notations utilisées pour exprimer des scénarios sont tout à fait informelles (langage naturel) et parfois semi-formelles (langage structuré). Les « use case » dans leur version originale [Jacobson 92] peuvent être considéré comme informels alors que les représentations graphiques introduites par [Andersson 95] peuvent être classées de semi-formelles. Il est souvent difficile de combiner la flexibilité de l'informel à la puissance du formel.

### C. Le niveau d'interactivité

L'interactivité se rapporte à la capacité qu'à l'utilisateur à progresser dans le scénario. Il peut être obligé de suivre le scénario du début à la fin ou il peut revenir en arrière, faire des branchements, parcourir une même partie une deuxième fois, etc.

Cette dimension dépend un peu du médium utilisé et du fait qu'il existe une machine facilitant la manipulation. Cependant, elle peut être vue comme une dimension à part entière dans le sens où un scénario sur un même médium, un niveau de formalité fixé, un niveau de détail donné et une structuration bien définie peut être présenté avec très peu d'interactivité ou avec beaucoup d'interactivité.

### D. Le niveau de détail

Au niveau du détail le plus bas, on dispose uniquement du nom du scénario et de rien d'autre. A l'opposé, chaque petite étape peut être décrite dans le moindre petit détail. Dans les deux cas, il peut s'agir du même scénario.

Pour trouver un niveau de détail adéquat, on peut procéder de la manière suivante. Un scénario peut être vu comme la description de la manière dont un ou plusieurs acteurs arrivent à un certain but. Le nom du scénario évoque ce but. Le but peut être raffiné en des sous-buts de plus en plus précis jusqu'au moment où un but correspond à une certaine tâche. Une tâche est caractérisée par sa manière exécutable, c'est-à-dire par le fait qu'elle peut être facilement décrite par une suite d'autres tâches plus élémentaires. On peut ainsi continuer la

décomposition jusqu'à un niveau de tâches élémentaires non décomposables. La Figure 22 montre une telle décomposition aussi appelée TKS[4] [IHM96].

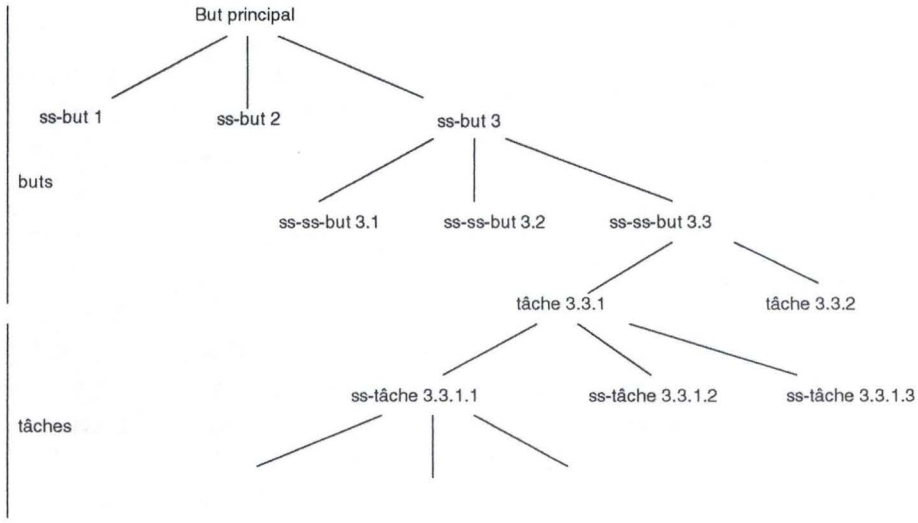



**Figure 22 décomposition en buts et tâches**

En allant du dessus vers la bas dans cette décomposition, nous ajoutons de plus en plus de détails au scénario. Voir un scénario à un certain niveau de détail signifie le lire à un certain niveau dans cette décomposition.

### E. La structuration

La décomposition d'un scénario comme elle est faite ci-dessus montre une certaine structuration interne. On peut aussi considérer une structuration externe qui concerne plutôt les relations entre différents scénarios.

Jacobson, par exemple, propose des « use case » abstraits et des « use case » d'extension comme mécanisme de structuration entre les « use case » [Jacobson 92]. Un « use case » abstrait est alors vu comme une partie commune à plusieurs autres « use case ». Un « use case » d'extension sert à distinguer les scénarios normaux (c'est-à-dire les plus fréquents) des scénarios exceptionnels.

Avoir une représentation sans structuration signifie qu'il n'existe pas de mécanismes créant des relations entre différents scénarios. A l'autre extrême, on peut trouver une structure du même type que celle utilisée lors de la structuration interne d'un scénario.

## 3.4.3 Les vues sur des scénarios

Les cinq dimensions détaillées ci-dessus sont relativement indépendantes. Cela signifie que la représentation d'un même scénario peut varier selon une dimension sans que les autres

[4] TKS = Task Knowledge Structure

dimensions doivent être adaptées. Cette propriété n'est pas tout à fait correcte puisqu'il est fort probable, par exemple, qu'une représentation hautement interactive soit assez bien structurée.

En se positionnant dans chacune de ces 5 dimensions, on détermine une vue sur les scénarios. Un même scénario peut donc être représenté suivant différentes vues. Chacune d'elle est plus ou moins intéressante selon la situation dans laquelle on se trouve.

## 3.5 Le contenu

Cette section essaye de clarifier un peu quelles informations exprimer dans un scénario. Pour cela, il se base surtout sur le rapport [CREWSD1-I1.1] qui distingue quatre aspects intéressants : le niveau d'abstraction du scénario, le contexte de celui-ci, la présence ou l'absence de certaines argumentations et la couverture du scénario.

### 3.5.1 Le niveau d'abstraction

Le niveau d'abstraction permet de faire la distinction entre des scénarios écrits au niveau type, instance ou un mélange entre les deux.

L'exemple suivant montre une phrase d'un scénario écrite d'abord de manière fort abstraite (dans le sens 'ne faisant intervenir que des types') et puis de manière de plus en plus concrète.

- Une voiture entre dans un carrefour.
- La voiture V entre dans le carrefour C.
- La voiture de monsieur X entre dans le carrefour Montgomery.
- La voiture de monsieur P Dupond entre dans le carrefour Montgomery par la voie sud.
- A 13h45, la voiture de monsieur P Dupond entre dans le carrefour Montgomery par la voie sud.

Cet exemple illustre plusieurs choses. Premièrement, il est difficile de définir des niveaux d'abstractions du scénario. Dire qu'il est du niveau type, instance ou un mélange entre les deux peut sembler insuffisant, vu toutes les possibilités qui existent. Et pourtant, déterminer uniquement la différence entre type et mélange entre les deux peut être difficile.

Deuxièmement, le niveau d'abstraction pourrait être rapporté aux objets décrits dans une phrase au lieu d'être rapporté à la phrase toute entière. Ainsi, c'est la voiture et le carrefour qui se précisent dans la phrase d'exemple ci-dessus.

Troisièmement, le niveau d'abstraction dépend du contexte dans lequel on travaille. Si nous considérons le problème de la modélisation d'un carrefour, 'voiture' et 'carrefour' sont des types. Si nous considérons la modélisation des actions possibles d'un véhicule, 'voiture'

serait une instance du même type que 'bus', tout comme le 'carrefour' serait une instance du même type que 'garage'.

« L'expérience montre que les personnes réagissent mieux à des scénarios concrets. » [CREWSD1-I1.1]. Un autre avantage des scénarios instanciés est qu'ils peuvent apporter des informations spécifiques qui, à priori, n'ont pas grand chose à voir avec le problème mais permettent de mieux comprendre l'environnement.

Leur désavantage est qu'ils sont fort coûteux. Un scénario instancié eut ajoute souvent des détails qui ne sont pas importants et sa taille est supérieure à la taille d'un scénario équivalent du niveau type.

### 3.5.2 Le contexte

Le contexte d'un scénario est, selon les membres de l'équipe CREWS, la portée du scénario par rapport à un système. Le scénario peut se rapporter uniquement au comportement interne du système, aux interactions entre le système et son environnement (ex. « use case »), au contexte organisationnel du système et même à l'environnement organisationnel.

Dans le cadre de l'exemple du carrefour où nous supposons qu'il sera réglé par un système considéré comme assemblage de composants matériels et logiciels, nous pouvons décrire un scénario montrant l'interaction entre ces différents composants suite à la détection d'une voiture devant un feu. Ce scénario se rapporte alors au comportement interne du système.

Si nous nous intéressons plutôt aux interactions entre les conducteurs des voitures et le système via les feux et les détecteurs, nous décrirons un scénario du deuxième type (interaction entre système et environnement).

« Les scénarios décrivant le contexte organisationnel du système peuvent être vus comme des explications de la connaissance du domaine d'application. » [CREWSD1-I1.1]. Si un scénario explique que des voitures ne passent pas au rouge, on peut dire qu'il est au niveau du contexte organisationnel du système. Le contexte organisationnel se rapporte aussi à la structure de l'entreprise, les responsabilités des personnes, leurs buts, etc.

L'environnement organisationnel est le contexte organisationnel poussé encore plus loin. Des scénarios contenant des informations sur l'environnement organisationnel décrivent non seulement l'environnement du système mais en plus l'environnement politique, géographique, social, etc. de l'entreprise voulant installer le système. Si un scénario décrit également le fait que la commission européenne impose tel et tel procédure à l'entreprise, alors on peu dire que le scénario contient une information de l'environnement organisationnel.

### 3.5.3 L'argumentation

Des argumentations concernant les fonctionnalités du système ou les raisons de certaines actions peuvent être inclues dans un scénario. L'équipe CREWS adopte le modèle IBIS distinguant :

- les positions : description de solutions alternatives concernant un problème,

- les arguments : arguments pour contredire ou supporter certaines positions,
- les enjeux : descriptions de problèmes ou conflits
- les décisions :choix d'un certaine position.

Un scénario peut ou ne peut pas contenir chacun de ces 4 types d'argumentations.

### 3.5.4 La couverture

La couverture d'un scénario correspond au type d'information que capte un scénario. L'équipe CREWS propose de distinguer les aspects fonctionnels, non fonctionnels et intentionnels.

Dans le domaine informatique, les aspects fonctionnels se rapportent généralement à la structure des données manipulées, les fonctions manipulant les données et le comportement d'un système.

Si un scénario s'occupe aussi d'aspects structurels (l'organisation des informations à manipuler), fonctionnels (les constructions permettant de manipuler les informations) et comportementaux, alors on dit que sa couverture est fonctionnelle. Dans le contexte de l'exemple du carrefour, un scénario expliquant l'apparence d'un feu (les 3 couleurs = structurel), la manière dont changent les couleurs du feu (fonctionnel) et le comportement des conducteurs face au système, il peut être classifié de fonctionnel.

On peut définir la description de l'objectif d'un scénario de deux manières. Premièrement, il peut s'agit du but poursuivi par celui qui construit le scénario. Ce but peut être du genre « décrire la manière habituelle de travailler ». Deuxièmement, l'objectif peut être celui d'un ou de plusieurs acteurs intervenant dans le scénario. Si ces acteurs représentent des personnes ou des rôles de l'entreprise, l'introduction de leurs but peut justifier beaucoup d'étapes dans le scénario.

Un scénario considéré d'avoir une couverture intentionnelle décrit les buts poursuivi, les problèmes rencontrés, les objectifs, les responsabilités, etc. de différents acteurs intervenant dans le scénario. Toujours sur base de l'exemple du carrefour, un scénario expliquant pourquoi le conducteur veut tourner à gauche a cette couverture intentionnelle.

Les aspects non-fonctionnels se rapportent plutôt aux éléments du type performance, contraintes temporelles, portabilité, contraintes budgétaires, documentation, etc. Un scénario faisant intervenir de tels éléments sera classifié parmi les scénarios non-fonctionnels.

## 3.6 Les problèmes liés aux scénarios

### 3.6.1 Le bon niveau de détail

Comme expliqué dans le paragraphe D, le niveau de détail est déterminé par l'endroit de lecture dans la hiérarchie de décomposition du scénario. Le problème consiste à savoir à quel endroit il faut se positionner.

Si le scénario ne contient que peu de détails, il arrive à décrire une situation assez complexe sans pour autant devenir illisible. S'il contient beaucoup de détails, le scénario aura une taille assez importante, ce qui entrave la facilité de lecture et de compréhension de celui-ci. Le seul moyen d'augmenter le niveau de détail sans perdre la lisibilité est de réduire la portée du scénario.

La détermination du bon niveau de détail dépend d'une part de la personne devant travailler avec ce scénario mais aussi du but du scénario. En effet, si le scénario doit décrire une fonctionnalité attendue du système, il est fort probable que le niveau de détail ne soit pas très élevé. Par contre, si le but est de découvrir la manière dont travaillent les utilisateurs, on peut s'attendre à un niveau de détail assez élevé.

A part ces quelques indications, il n'existe aucune règle permettant de dire dans quelles situations il faut mettre quels niveaux de détail. Un problème majeur pour des analystes débutants est de trouver la forme d'expression adaptée à la situation rencontrée.

### 3.6.2 Le bon niveau d'abstraction

Dans la section 3.5.1, nous avons donné un petit aperçu concernant les problèmes liés au niveau d'abstraction. Faut-il raisonner au niveau instance ou au niveau type ?

Comme pour le niveau de détail, le but du scénario joue un rôle essentiel dans la détermination du niveau d'abstraction. Plus le niveau d'abstraction est élevé, plus le scénario permet d'exprimer des choses globales. Dans le cadre d'une détermination d'un comportement générique, il est donc intéressant de rester à un niveau d'abstraction élevé. Lorsqu'il s'agit de découvrir des fonctionnements parfois subtils de l'environnement du système, il est plus efficace de réduire le niveau d'abstraction pour raisonner au niveau instance.

On peut aussi ajouter à cela qu'en fonction de sa formation, une personne devant travailler avec des scénarios aura tendance à utiliser des expressions plus ou moins abstraites. Sans vouloir apporter un jugement, il est fort probable qu'un ouvrier décrit sa manière de travailler par des scénarios concrets alors qu'un chef de département le fera de manière plus abstraite.

Ici aussi, le problème est le même :comment trouver la forme d'expression la plus adaptée à une situation donnée ?

### 3.6.3 L'informel, le semi-formel ou le formel ?

Lors de la discussion des problèmes rencontrés dans l'ingénierie des besoins (paragraphe 2.6), la question du choix d'expression sous forme formelle ou non formelle reste importante. Cette question est fortement liée au problème de la lisibilité d'un scénario.

En général, un texte écrit dans la langue naturelle est compréhensible par plus de personnes qu'un texte écrit de manière formelle ou mathématique. Cependant, la représentation graphique apporte souvent un plus au niveau formel tout en restant très lisible. Le problème est alors que tout ne peut être exprimé. En langue naturelle, des contenus de type argumentation (section 3.5.3) peuvent être introduits sans problème. Sur un graphique dessiné avec des formes bien définies, ces éléments doivent être ajoutés de manière informelle (texte libre noté à côté).

Il est aussi important d'avoir un langage qui soit proche de la manière dont s'exprime le client. L'effort de traduction ne peut être trop important. La langue naturelle non structurée ne demande aucun effort puisqu'il suffit de noter directement ce que dit le client. Un langage formel pose un peu plus de problèmes à ce niveau. Il faut donc veiller à retrouver les mêmes constructions dans le langage d'expression de scénarios que dans l'expression du client.

Le problème consiste donc à trouver un bon compromis entre la lisibilité, l'expressivité, la facilité de traduction et la possibilité de traitement automatique obtenu grâce à des notations formelles.

## *3.7 Exemples de scénarios*

Nous avons trouvé dans la littérature différents exemples de scénarios, par exemple dans [SEED 95], [Jacobson 92], [Regnell 95] et [CREWSD1-I1.1]. A côté d'une sélection de ceux-ci, nous allons donner quelques exemples de scénarios se rapportant à l'étude de cas du carrefour.

### 3.7.1 Seed-Pro

SEED (« Software Environment for the Early Phases of Building Design ») est un projet à la Carnegie Mellon University à Pittsbrugh. Le but de ce projet est de développer un programme architectural (« architectural program ») ayant pour but d'initier un processus de design. Le document [SEED 95] décrit les fonctionnalités que le système (Seed-Pro) doit offrir d'un point de vue utilisateur.

Le document identifie et décrit 30 « use case ». Pour eux, un scénario est la mise en commun de plusieurs « use case ». Ainsi, le scénario « Ecrire un programme à partir de rien » est constitué des « use case » suivants :

1. « démarrer SEED-Pro »,
2. « démarrer un nouveau projet »,
3. « introduire les spécifications du projet »,
4. « introduire les spécifications du site »,
5. « introduire les spécifications de construction »,

6. « introduire les spécifications du profil du client »,
7. « introduire les spécifications du budget »,
8. « attacher un FU à un SU[5] »,
9. « créer (ajouter) une instance d'un FU »,
10. « modifier une instance de FU/SU »,
11. « créer une alternative de programmation »,
12. « agréger des instance de FU »,
13. « sauvegarder la programme vers la base de donnée »,
14. « sortir du programme ».

Nous allons donner les détails des 3 premiers « use case ». Remarquons qu'il s'agit de scénarios travaillés servant de cahier des charges pour ce projet. On trouve également une représentation graphique associée au texte et des informations supplémentaires comme les pré- et postconditions, les objets intervenants, les besoins spécifiques, etc.

Les trois paragraphes suivantes sont des traductions libres des paragraphes correspondant à ces 3 « use case » dans le document [SEED 95].

**« Démarrer SEED-Pro. »**

Le flux des événements débute lorsqu'un designer sélectionne l'option « Program : New » ou « Open » de l'écran principal SEED. Le système affiche alors l'écran Projet. Le use case se termine lorsque cet écran est affiché.

**« Démarrer un nouveau projet. »**

Le flux des événements pour ce use case peut être débuté par deux moyens différents. Premièrement, l'utilisateur peut choisir « New » du menu Projet de l'écran principal SEED. L'autre possibilité est de choisir « New » du menu « file » de l'écran projet.

Ensuite, le système demande au designer d'entrer un nom pour le projet. Le designer tape le nom dans la boite de dialogue proposée par le système et confirme.

L'écran 'projet' du nouveau projet est affiché avec le nom du projet s'affichant dans le champ adéquat de cette fenêtre. C'est considéré comme étant le projet courant et le designer peut maintenant commencer à éditer toutes les versions de ses programmes. Toutes les fenêtres appartenant au projet précédent sont fermées et une icône représentant le projet précédent apparaît dans l'écran « Session Manager » montrant qu'il n'est plus le projet courant. L'utilisateur peut activer n'importe quel projet chargé précédemment dans le projet courant en sélectionnant son icône dans l'écran « Session Manager » et en disant au système de rendre le projet courant.

**« Introduire les spécifications du projet. »**

Le flux des événements commence lorsque le designer sélectionne l'option 'spécifications' de l'écran 'projet'. Le système affiche alors l'écran de spécification. L'écran comporte 5 options différentes permettant au designer d'introduire des spécifications relatives

---

[5] FU = functional unit ; SU = specification unit

à la construction, au site, au budget, à la planification de l'implémentation et au profil du client. Le « use case » se termine lorsque l'écran de spécification est affiché.

### 3.7.2 Regnell et al.

Dans leur document [Regnell 95], Björn Regnell, Kristofer Kimbler, Anders Wesslén décrivent 3 « use case » d'exemples. Nous en reproduisons un ici. Il est écrit en langue naturelle fortement structurée et une représentation graphique correspondante existe. Nous ne donnerons que la partie textuelle.

Dans ce cas, la structuration de la langue naturelle comporte plusieurs aspects.

- Il existe des rubriques annexes au vrai scénario (la suite des actions). Ces rubriques sont une précondition, les conditions de bon déroulement (aux points de décisions on prend le bon cheminement) et un postcondition.
- Il n'y a qu'une seule phrase par ligne.
- Chaque phrase est muni d'un numéro.
- Les attributs sont mis en italique.
- Les objets sont mis en gras.

L'exemple est pris du cadre d'une spécification d'un ATM[6]. Un ATM est un dispositif permettant de régler un certain nombre d'opérations bancaires comme, par exemple, le retrait d'argent. L'exemple ci-dessous étudie cette opération.

**IC** **Précondition**
IC.1 Le système est prêt pour des *transactions*

**FC** **Conditions de bon déroulement**
FC.1 La **carte** de l'utilisateur *est valide*
FC.2 L'utilisateur entre un **code** *valide*
FC.3 L'utilisateur entre un **montant** *valide*
FC.4 La machine dispose du **montant** *demandé*

**FE** **Flux des événements**
FE.1 L'utilisateur introduit la **carte**
FE.2 Le système *vérifie si* la **carte** *est valide*
FE.3 Le système *demande* le **code**
FE.4 L'utilisateur introduit le **code**
FE.5 Le système *vérifie si le* **code** *est valide*
FE.6 Le texte *« introduire montant ou choisir balance »* est affiché
FE.7 L'utilisateur introduit le *montant*

---

[6] ATM = Automated Teller Machine

FE.8 Le système *vérifie si le montant est valide*
FE.9 Le système *prépare* **l'argent**
FE.10 **L'argent** *est éjecté*
FE.11 *Le texte « retirer l'argent »* est affiché
FE.12 L'utilisateur retire **l'argent**
FE.13 La **carte** *est éjectée*
FE.14 *Le texte « retirer la carte »* est affiché
FE.15 L'utilisateur prend la **carte**
FE.16 Le système *prépare les informations* pour **le reçu**
FE.17 Le **reçu** *est imprimé*
FE.18 *Le texte « retirer le reçu »* est affiché
FE.19 L'utilisateur prend le **reçu**

**TC** **Postcondition**
TC.1 Le système est prêt *pour des transactions*

### 3.7.3 Crews

Le document [CREWSD1-I1.1] proposant un cadre de classification des scénarios cite quelques exemples de scénarios. Ces exemples sont tirés des sources de l'équipe ayant réalisé le document [CREWSD1-I1.1]. Nous en avons choisi deux : un premier tiré de leur référence [Carroll 95][7] représente une partie d'un scénario exprimé sous forme textuelle en langage naturel non structuré, et le deuxième tiré de leur référence [Potts et al., 1994][8] représente un scénario de planification de réunions exprimé sous forme tabulaire.

A nouveau, les deux scénarios exprimés ci-dessous sont des traductions libres de textes trouvés dans [CREWSD1-I1.1].

#### A. Carroll

La seule règle à suivre pour créer un tel scénario en langue naturelle non-structurée est d'appliquer les consignes générales d'écritures de textes : une idée par paragraphe, phrases pas trop longues, style simple et facile à lire, etc.

« Harry, un créateur de curriculum, a juste rejoint un projet développant un système d'information multimédia pour enseigner l'ingénierie. Il parcourt l'historique du projet sur vidéo. Des ensembles de clips sont catégorisés sous des rubriques principales présentées sur base d'icônes : sous certaines de ces rubriques, on trouve des sous-rubriques qui peuvent être atteintes par un système de menu.

---

[7] [Carroll 95] J.M. Carroll, « The Scenario Perspective on System Development », in Scenario-Based Design :Envisioning Work and Technology in System Development, Ed J.M. Carroll, 1995.

[8] [Potts et al., 1994] C. Potts, K. Takahashi, A. I. Anton, « Inquiry-based Requirements Analyses », in IEEE Software, Vol 4 No 1, 1987.

Il sélectionne l'icône du designer Lewis, l'icône de visualisation des enjeux et un horaire du début du projet. Il sélectionne ensuite « play clip » et voit une scène brève dans laquelle Lewis décrit sa vue du projet comme rendant possible un nouveau monde d'une éducation basée sur la collaboration et l'expérience.

... »

### B. Potts

Comme chez Regnell ci-dessus (3.7.2), les différentes étapes sont numérotées. On trouve cependant une structure supplémentaire qui consiste à mettre en évidence l'acteur responsable de chaque action. Le tableau ne reflète pas d'autres particularités structurelles.

| No | Agent | Action |
|---|---|---|
| 1 | Initiateur | Demande une réunion d'un certain type avec des informations sur la réunion (par exemple agenda / but) et des dates possibles. |
| 2 | Planificateur | Ajoute les participants (actifs / importants) par défaut, etc. |
| 3 | Initiateur | Détermine trois participants. |
| 4 | Initiateur | Identifie un présentateur comme participant actif. |
| 5 | Initiateur | Identifie le supérieur de l'initiateur comme participant important. |
| 6 | Initiateur | Envoie la demande des préférences. |
| 7 | Planificateur | Envoie par courrier les messages appropriés aux participants (incluant des demandes particulières pour le supérieur et le présentateur). |
| 8 | Participant ordinaire | Répond avec un ensemble d'exclusions et de préférences. |
| 9 | Participant actif | Répond avec un ensemble d'exclusions et de préférences ainsi que des besoins en équipement. |
| 10 | Planificateur | Demande l'équipement nécessaire. |
| 11 | Participant important | Répond avec un ensemble d'exclusions et de préférences ainsi que des préférences sur la localisation. |
| 12 | Planificateur | Planifie la réunion en se basant sur les réponses, les politiques et la disponibilité des lieux. |
| 13 | Planificateur | Envoie des messages de confirmation à tous les participants ainsi qu'à l'initiateur. |

## 3.7.4 Somé

Le document [Some et al] distingue la partie description du domaine d'application de la partie scénario. Cette distinction est comparable à la distinction entre le vocabulaire utilisé pour écrire le scénario et le scénario lui-même.

On trouve dans ce document un exemple de chaque partie : d'abord la déclaration du vocabulaire du domaine d'application et ensuite un scénario écrit en se basant sur ce vocabulaire.

Nous reproduisons ici uniquement la partie scénario. Nous ne l'avons pas traduit pour ne pas perdre la structure existante. Le scénario est écrit en langue naturelle fortement structurée.

when USER_A *handset* is *down*

if USER_A *pick_ups handset* then CONTROLLER *sends* USER_A *tone*

before 20 sec if USER_A *dials* USER_B then CONTROLLER *checks* USER_B *status*

if USER_B *status* is *idle* then CONTROLLER *rings* USER_B

if USER_B *pick_ups handset* then CONTROLLER *establishes A_B_comm transition delay* 40 sec *on expiry* CONTROLLER *stop_A* and CONTROLLER *sends* USER_A *tone*.

# 4. Des scénarios pour ALBERT

## 4.1 Introduction

Le but de cette section est de discuter d'un langage d'expression de scénarios lié au langage ALBERT. Ce langage incorpore les éléments bénéfiques des scénarios et essaye d'éviter les problèmes liés à ceux-ci. Il est divisé en 2 parties : l'expression du contenu du scénario et l'expression du contexte du scénario. Au niveau du contexte, nous distinguons une partie informelle et une partie formelle.

Un utilisateur expliquant sa manière de travailler ne se contente pas uniquement de décrire les étapes qu'il doit suivre pour réaliser sa tâche. Il donne des explications supplémentaires liées aux buts de sa tâche, à la raison de certains choix pris au cours du travail, aux situations dans lesquelles il réalise cette tâche et aux situations dans lesquelles il ne la réalise pas. Toutes ces informations font partie de ce que nous appellerons le contexte d'un scénario.

Le contenu est la partie 'exécution' du scénario. C'est le contenu qui exprime la suite des actions et la communication entre les différents acteurs intervenant dans le scénario. Tout ce qui n'est pas contenu sera classé dans le contexte.

La distinction entre le contenu et le contexte d'une scénario est importante dans la spécification d'un langage d'expression de scénarios. La nature des informations est assez différente : le contenu s'exprime plutôt sous une forme opérationnelle, c'est-à-dire étape par étape, le contexte par contre s'exprime plutôt de manière déclarative. Il est donc intéressant de définir des langages différents, l'un pour le contenu et l'autre pour le contexte. En effet, la manipulation du contenu peut être améliorée grâce à un langage formel. Cette manipulation peut être très importante lorsque le scénario sert d'entrée pour une animation, par exemple. Le contexte, et surtout la partie description des buts, des objectifs, des personnes intervenant dans l'élaboration, etc., est beaucoup plus variable. Il sert surtout à la classification du scénario et à la structuration entre scénarios. Il demandera donc un langage nettement plus flexible et moins formel.

Le contexte a deux buts. Un premier est de fournir des informations supplémentaires sur la création du scénario, de ses buts, de ses particularités, etc. Grâce à ces informations supplémentaires, l'analyste aura des facilités pour la manipulation des scénarios lors de grands projets.

Le deuxième but est de décrire précisément une ou des préconditions, une ou des postconditions et le fonctionnement des éléments de l'environnement auxquels le scénario ne s'intéresse pas.

La précondition décrit soit un cadre dans lequel le scénario va se dérouler et sera un scénario valide, soit cadre dans lequel le scénario ne peut pas se dérouler. Elle est assez indépendante du contenu du scénario. On peut très bien avoir plusieurs préconditions

différentes pour un même scénario ou appliquer la même précondition à des scénarios différents. Dans le cadre du problème du carrefour, le description du carrefour concret considéré pour un scénario fait partie d'une telle précondition. Elle peut être utilisée pour plusieurs scénarios différents. D'un autre côté, il est intéressant de donner un scénario de base comme par exemple une voiture qui traverse le carrefour tout droit et de l'utiliser dans le cadre de différents carrefours concrets ou un même carrefour soumis à des charges différentes.

La postcondition donne des indications sur la suite du déroulement de l'action. Il peut être intéressant de pouvoir dire qu'après le déroulement d'un certain scénario, telle ou telle condition sera toujours vraie.

La description du comportement de l'environnement pendant l'exécution du scénario devient très intéressante lorsque le scénario est utilisé dans le cadre d'une animation. Elle permet de définir sous une forme assez compacte la manière dont un animateur (logiciel) doit traiter une multitude d'indéterminations. Pour le carrefour, il peut être intéressant de dire qu'il y a, en moyenne, une voiture toutes les x secondes arrivant d'une certaine bande alors qu'on ne s'intéresse pas du tout à cette bande.

Dans une première partie, nous allons faire l'inventaire de ce qui est utilisé actuellement comme langage d'expression du contenu et du contexte de scénarios dans la littérature (section 4.2). Chaque langage sera accompagné d'une petite critique. Ensuite, nous allons définir un langage adapté à l'expression de scénarios dans le contexte du langage ALBERT (section 4.3). Cette deuxième partie débutera par un rappel des éléments importants d'un langage de scénario (section 4.3.1) et les particularités des scénarios pour ALBERT (section 4.3.2). Ensuite nous définirons un langage d'expression du contenu (section 4.3.3) et deux langages d'expression du contexte de scénarios (section 4.3.4). Nous terminerons par un exemple complet utilisant ces langages (section 4.3.6).

## 4.2 Etat de l'art

Cette section tente de faire l'inventaire des formalismes les plus utilisés pour exprimer des scénarios. Elle est divisée en une partie présentant les langages d'expression de contenu et une deuxième présentant des langages d'expression de contexte.

Le but n'est pas d'être exhaustif mais plutôt de donner un bon aperçu des différents formalismes rencontrés.

Pour les langages exprimant le contenu, nous donneront également une petite critique par rapport à leur utilisation dans le contexte de scénarios ALBERT. L'ensemble de ces critiques sera repris dans un tableau récapitulatif.

### 4.2.1 Les langages du contenu

Pour cette partie, nous avons choisi les formalismes suivants : message sequance charts (MSC), diagrammes état-transition, réseaux de Pétri, diagrammes de flux et le langage naturel structuré.

Les MSC ont été choisi parce qu'ils nous serviront dans la définition du langage pour les scénarios ALBERT.

Les diagrammes état-transition sont intéressants grâce à leur simplicité et à leur possibilité de structuration. En plus, les vies en ALBERT sont une sorte de diagramme état-transition.

Les réseaux de Pétri sont intéressants pour leur aspect formel et pour la facilité de les exécuter.

Les diagrammes de flux dans leur version de [Bodart 93] ont l'intérêt d'être adaptés à l'ingénierie des besoins. Ils peuvent être considérés comme les prédécesseurs des scénarios.

Le langage naturel est le moyen d'expression de scénarios le plus utilisé actuellement dans les entreprises. Structuré ou non, il reste le moyen d'expression le plus riche.

Les autres langages rencontrés dans la littérature sont en général une adaptation d'un ou de plusieurs de ces formalismes. Par exemple, pour la représentation du contenu d'un scénario, les « use case » de Jacobson sont une combinaison de langue naturelle et de MSC.

## A. Les message sequence charts

« L'organisation de standardisation des télécommunications ITU-T a présenté la recommandation Z.120 [ITU-MSC 93][9] en mars 1993 avec le but 'de mettre à disposition un langage de trace pour la spécification et la description du comportement des communications entre composants d'un système et leur environnement par le biais d'échanges de messages.' [...]

MSC montrent l'envoi de messages entre instances. Une instance peut être un utilisateur, l'environnement en général, le système, une partie du système ou un autre système.

Un avantage majeur avec les « message sequence charts » est qu'ils sont intuitifs. Ils peuvent être compris par presque tout le monde. Un autre avantage est le travail de standardisation par ITU. Ce standard est mis à jour régulièrement pour s'adapter aux besoins des utilisateurs » [Andersson 95].

Il existe deux formes de représentations des MSC : une graphique et une textuelle.

La représentation graphique reprend dans l'axe des X toutes les instances des acteurs intervenants dans le scénario. L'axe des Y reprend le temps qui augmente de haut en bas. Une ligne du temps est associée à chaque instance d'acteur. L'échange des messages est représenté par des flèches partant de la ligne du temps d'un acteur pour rejoindre la ligne du temps de l'autre acteur. Cette flèche peut être complétée par le nom du message échangé. La Figure 23 montre un exemple d'un tel diagramme. L'exemple montre la version UML [UML] des MSC qui introduit déjà une première extension : les élargissements des lignes verticales à certains endroits. Nous y reviendrons dans la section détaillant les MSC (B).

---

[9] ITU, *Message Sequence Chart* (MSC), 1993

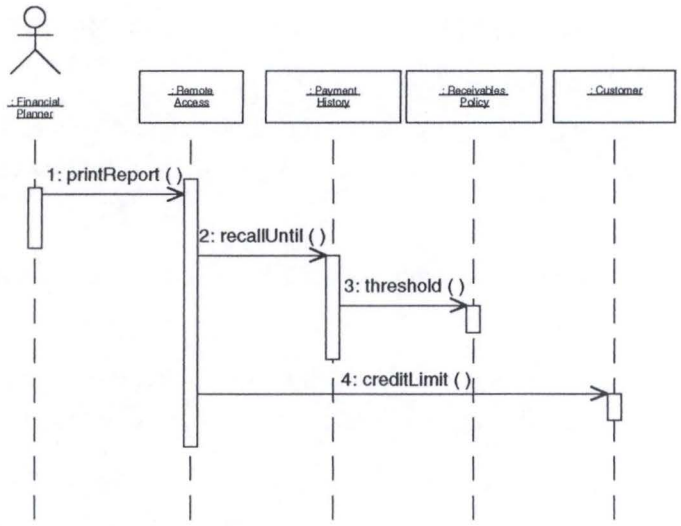



**Figure 23 - exemple de diagramme MSC[10]**

Il existe plusieurs extensions à cette notation permettant entre autre d'inclure des répétitions, des conditions (branchements), des aspects temps-réel, des exceptions, etc. Nous allons y revenir dans la section réservée uniquement aux MSC.

Comme déjà dit ci-dessus, les MSC sont largement utilisés. C'est un avantage important puisqu'une large utilisation assure la remise en question et l'adaptation du formalisme aux nouveaux besoins.

Beaucoup d'auteurs s'intéressent aux MSC offrant une notation formelle. Dans le contexte des scénarios ALBERT, c'est surtout les travaux sur les extensions temporelles [Abdallah 97] qui intéressent beaucoup. C'est aussi l'identification aisée des acteurs qui peut apporter un grand avantage dans le contexte des spécifications en ALBERT. Une instance d'un agent ALBERT correspond à une instance d'un acteur en MSC.

Un autre avantage très important est sa sémantique assez immédiate. Comme dit plus haut, presque tout le monde peut lire et comprendre la représentation graphique d'un « message sequence chart ».

Un des principaux concepts des MSC est celui du message. Le problème est que ce concept est inexistant (en tout cas sous cette forme) dans le vocabulaire utilisé dans une spécification en ALBERT.

Un autre problème est que les MSC, en tout cas dans leur version de base, ne peuvent exprimer que des scénarios complètement instanciés. Or cette caractéristique est trop restrictive par rapport aux besoins des scénarios ALBERT.

---

[10] Ce dessin est tiré d'un exemple livré avec la version de démonstration de l'outil case Rational Rose implémentant la notion UML. Pour plus d'informations, le lecteur pourra consulter le site Internet http:\\www.rational.com.

## B. Les diagrammes état-transition

Les diagrammes état-transition sont très bien connus dans le domaine informatique. Leur concepts de base sont les états et les transitions. Un état représente une situation stable dans laquelle un acteur ou tout un système se trouve. Les transitions correspondent à des événements faisant passer le système d'un état à un autre.

Les états sont représentés par des rectangles et les transitions par des flèches entre ces rectangles. On identifie souvent un état initial servant de point de départ. La Figure 24 montre un exemple de diagramme état-transition.

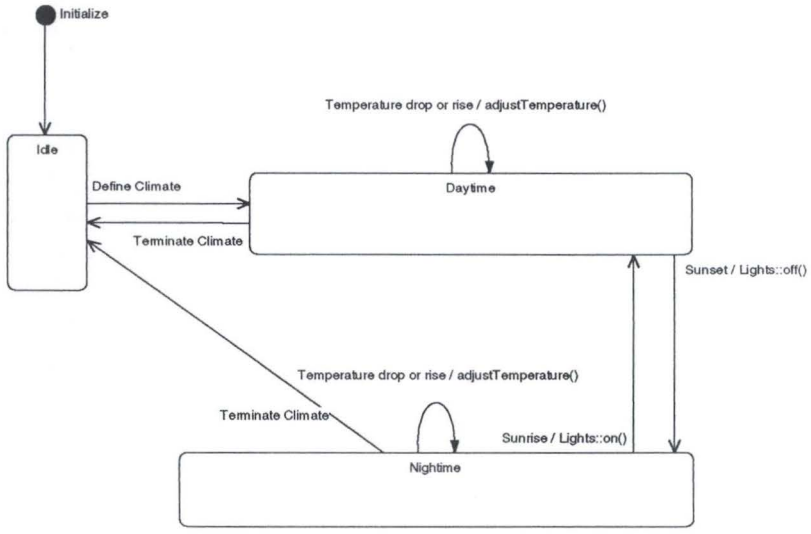



**Figure 24 - exemple de diagramme état-transition**[11]

Certaines extensions existent par rapport à ces diagrammes. Ainsi, on peut associer des durées aux transitions ou même des points de décision lors d'une transition. On peut également associer des actions aux transitions. Des diagrammes de ce type peuvent être incorporés l'un dans l'autre permettant ainsi de réduire la complexité. Les langages de design ROOM [Selic 94] et STATEMATE exploitent beaucoup ces extensions.

L'avantage des diagrammes état-transition est qu'ils sont largement utilisés et déjà depuis assez longtemps. Ils ont souvent prouvé leur utilité, leur flexibilité et leur robustesse. Leur sémantique est bien connue et ils sont, tout comme les MSC, facilement lisibles par presque tout le monde.

Un autre avantage dans le contexte d'un scénario ALBERT est que les vies correspondants à la spécification d'un agent ne sont rien d'autres que des diagrammes état-transition un peu spéciaux. En effet, une vie est représentée comme une suite d'états changeant uniquement lors d'occurrences d'actions.

---

[11] Ce dessin est tiré d'un exemple livré avec la version de démonstration de l'outil case Rational Rose implémentant la notion UML. Pour plus d'informations, le lecteur pourra consulter le site Internet www.rational.com.

Malgré cela, il ne faut pas oublier que les scénarios mettent l'accent sur la suite des actions et non sur la suite des états. En plus, un élément important des scénarios, à savoir la communication entre acteurs ne peut être représentée immédiatement sur des diagrammes état-transition. On peut encore ajouter à cela qu'en général, on ne voit pas immédiatement l'exécution d'un scénario sur un tel diagramme. Sauf association d'un ordre de parcours des transitions, un diagramme état-transition n'exprime pas directement un scénario.

## C. Les réseaux de Petri

Les réseaux de Pétri sont fort connus parmi les mathématiciens et informaticiens. Il s'agit d'une sorte de diagramme état-transition particulier pouvant exprimer certaines conditions directement.

Il est constitué d'un ensemble d'états (généralement représentés sous forme de cercles) et de transitions (représentées sous forme de barres) reliés entre eux par des flèches. Les états sont marqués, ce qui veut dire qu'on leur associe un certain nombre de jetons. Les flèches sont libellées par des nombres. Le nombre indiqué sur une flèche allant d'un état vers une transition représente une précondition. Elle est remplie si le nombre associé à la flèche est plus grand que le nombre associé à l'état. Dans ce cas, la transition peut être 'tirée'. Une flèche allant d'une transition vers un état signifie que si la transition est tirée, alors il faut augmenter le nombre associé à l'état du nombre associé à la flèche.

Sur la Figure 25, la transition t1 peut être tirée. Le résultat est représenté dans la Figure 26.

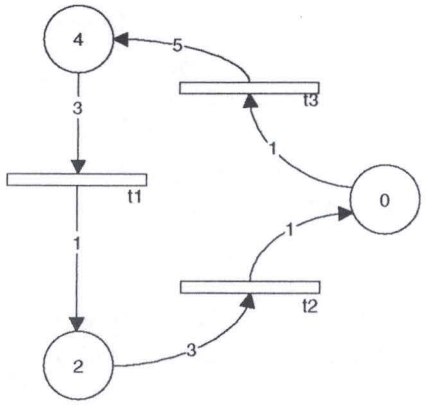



**Figure 25 - exemple de réseau de Petri (avant le tir de la transition t1)**

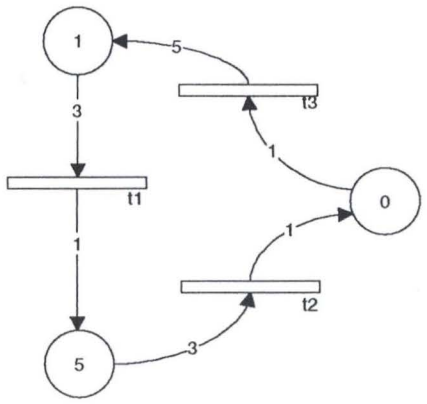



**Figure 26 - exemple de réseau de Petri (après le tir de la transition t1)**

Il existe différentes extensions aux réseaux de Petri concernant entre autre le problème de la complexité des diagrammes lors de la modélisation de situations fort compliquées et introduisant des formalismes liés aux contraintes temps-réel.

Les avantages et désavantages des réseaux de Petri sont les mêmes que ceux des diagrammes état-transition expliqués ci-dessus (section B). Ils expriment un comportement, mais pas sous la forme la plus intéressante, pour représenter des scénarios.

## D. Les diagrammes de flux

Le but initial d'un diagramme de flux est d'exprimer l'acheminement de messages entre différents acteurs. Ce qu'ils ont en plus par rapport aux MSC, c'est la représentation des traitements de ces messages. Le concept de traitement est comparable à celui d'action dans le cadre des scénarios. L'arrivée de messages provoque l'exécution d'une action et la terminaison de celle-ci génère en général de nouveaux messages.

La représentation ressemble un peu à celle des MSC. Les acteurs intervenant dans le scénario sont représentés selon l'axe des X. Le temps est représenté selon l'axe des Y de haut en bas. La Figure 27 montre un exemple de diagramme de flux.

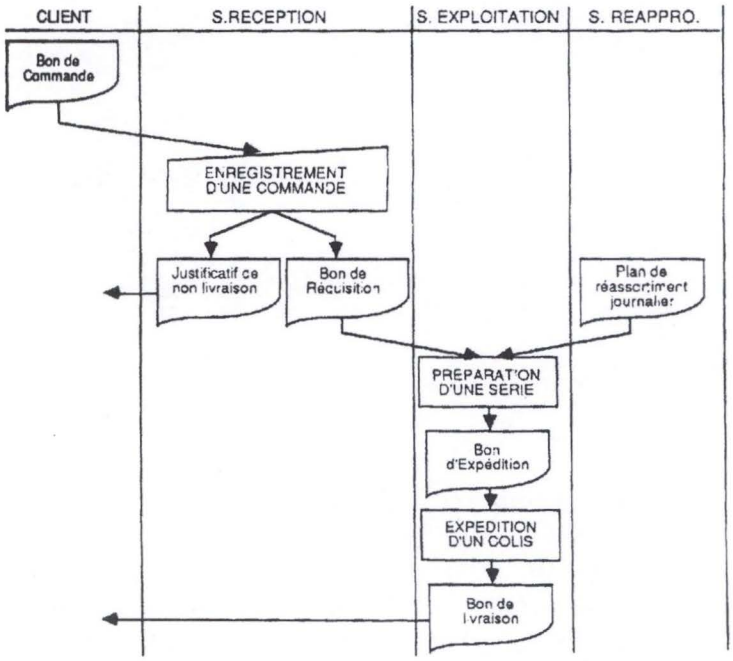



**Figure 27 - exemple de diagramme de flux [Bodart 93]**

Les critiques applicables à ce genre de diagramme pour la représentation de scénarios sont fort semblables à celles énoncées pour les MSC (section 4.2.1). Il s'agit principalement du problème lié à la notion de messages inexistante dans le vocabulaire d'ALBERT.

## E. Le langage naturel structuré

Le langage naturel structuré est un texte libre qui respecte une certaine structuration. La structuration comporte en général la division en phrases relatives à des actions élémentaires ainsi que la numérotation de ceux-ci (cfr. [Regnell 95]). Dans certains cas, des mots importants sont mis en évidence et les phrases sont toutes construites de la même manière.

Voici un scénario écrit en langage naturel structuré.

1. Je m'*approche* du **carrefour**. Sur ma **bande** il n'y a personne et le **feu** est **vert**.
2. Je *rentre* dans le **carrefour**. Sur ma **trajectoire** se trouvent déjà 4 **voitures**.
3. Les **voitures** d'en face *arrivent* sans interruption et on est **bloqué**.
4. Lorsque le **feu** d'en face *devient* **rouge**,
5. les **voitures** devant moi *quittent* le **carrefour**.
6. Je *sors* du **carrefour** au moment où les **voitures** à ma gauche et à ma droite peuvent rentrer dans le **carrefour**.

Par ligne il n'y a qu'une seule action importante dans le scénario. Les noms des actions sont écrits en italique et les agents ou propriétés importantes des agents sont écrits en gras. Ceci est une possibilité parmi plusieurs de structurer un scénario écrit en langue naturelle.

L'avantage de cette méthode est sa flexibilité et son expressivité. Un scénario écrit en langage naturelle peut exprimer sans beaucoup d'effort n'importe quelle situation. Le fait de structurer ce texte augmente sa lisibilité.

Son désavantage principal est son aspect peu formel. Cela provoque d'une part la difficulté de proposer une représentation graphique équivalente et d'autre part la difficulté de réaliser des vérifications et dérivations automatiques à partir des scénarios.

## F. Récapitulatif

Le tableau suivant reprend les principales critiques liées aux différentes représentations de scénarios introduites ci-dessus. Rappelons que ces critiques se rapportent à l'adéquation des représentations aux scénarios ALBERT, qui, comme on l'a vu au paragraphe 4.3.2, sont un peu particulières.

Le Tableau 1 reprend les principaux arguments pour et contre un certain langage.

| *Représentation* | *Critique positive* | *Critique négative* |
|---|---|---|
| MSC | • sémantique intuitive<br>• efforts de standardisation<br>• beaucoup utilisé<br>• extensions temp-réel | • concept de message inadéquat pour ALBERT<br>• version de base n'admet pas de boucles, points de décision, ... |
| Etat-transition | • beaucoup utilisé<br>• sémantique connue<br>• facilement lisible<br>• vie ALBERT = état-transition | • il faut une suite d'actions et pas une suite d'états<br>• ne représente pas la communication entre acteurs<br>• ne montre pas le déroulement d'un scénario |
| Réseau de Petri | idem état-transition | idem état-transition |
| Diagramme de flux | idem MSC | idem MSC (problème des messages) |
| Langage naturel structuré | • flexibilité<br>• structure apporte une bonne lisibilité | • peu formel<br>• représentation graphique équivalente difficile à trouver |

**Tableau 1 - tableau récapitulatif des langages de scénarios**

### 4.2.2 Les langages du contexte

Comme expliqué plus haut, nous distinguons deux buts pour les langages décrivant le contexte de scénarios. Le premier est de décrire une mise en situation intéressante pour l'exécution d'un scénario. Le deuxième est de retenir des informations relatives à la création et à la gestion des scénarios.

L'état de l'art ci-dessous ne fera pas la distinction entre ces deux notions. Cela est surtout dû au fait que les langages existant pour représenter le contexte d'un scénario les mélangent.

Nous nous sommes basés sur les 4 documents suivants : [SEED 95], [Vemulapalli], [Regnell 96] et [CREWSD1-I1.1].

Les deux premiers décrivent des modèles concret (appelés « template ») pour la représentation de « use case ». Nous les avons choisis parce qu'ils sont issus d'expériences concrètes.

Le troisième nous montre un méta-modèle associé à la notion de « use case » proposant des éléments de contexte. C'est grâce à son méta-modèle que nous l'avons choisi.

Le dernier nous intéresse par son approche théorique. Le but de ce document étant de classifier les scénarios, il analyse bien les différents éléments importants relatifs au contexte.

#### A. SEED

Dans le cadre du projet SEED (Software Environement for the Early Phases of Building Design), il existe un document définissant les besoins du logiciel SEED-Pro grâce à 30 scénarios [SEED 95]. Le document ne donne pas uniquement la suite des actions qui constituent les scénarios mais aussi des informations supplémentaires. Les informations concernant un scénario sont groupées sous les rubriques suivantes.

- « Interaction diagram » : il s'agit d'une représentation du scénario sous forme d'un bMSC.
- « Flow of events » : c'est la version textuelle du scénario.
- « Associations » : cette rubrique donne les structurations entre scénarios. Les associations possibles sont l'utilisation et l'extension.
- « Participating Objects » : rappelle les acteurs faisant partie du système et intervenants dans le scénario.
- « Special Requirements » : contient des informations supplémentaires relatives à la spécification.
- « Pre-conditions » : préconditions.
- « Post-conditions » : postconditions.

- « Notes » : remarques supplémentaires.

Certaines rubriques ne sont pas obligatoires. Ainsi, la rubrique « special requirements » est vide dans 29 des 30 scénarios. Les pré- et postconditions n'existent pas pour quelques scénarios et les notes n'apparaissent qu'une seule fois.

## B. WorldCom

Le document [Vemulapalli] intitulé « A Use Case FAQ » donne outre des réponses à des questions concernant les « use case », un modèle (template) d'écriture de « use case »[12]. Ce modèle comporte également un ensemble de rubriques qui ne sont pas directement liées à la suite des actions d'un scénario. Voici la liste des rubriques rencontrées.

- « Use Case » : nom du « use case ».
- « Actors » : tous les acteurs et/ou autres systèmes initiant le « use case ».
- « Contact Person / Dept » : le nom et le département de la personne qui a fourni les informations relatives au « use case ».
- « Name » : nom de la personne ayant écrit le « use case ».
- « Phone » : son numéro de téléphone.
- « Summary » : une très brève description du contenu du « use case ».
- « Pre-conditions » : les conditions qui doivent être remplis pour que le « use case » peut être exécuté.
- « Post-conditions » : l'état du système après l'exécution du « use case ».
- « Exceptions » : toutes les possibilités alternatives pouvant être parcourues pour des raisons diverses. Par exemple, lorsque les préconditions ne sont pas satisfaites.
- « Security Exceptions » : toutes les considérations de sécurité devant être implémentées durant le développement du « use case ».
- « Related Use Cases » : la liste des autres « use cases » en rapport avec celui-ci.
- « Attachements » : tout document ou référence supplémentaire devant être spécifié pour ce use case. Par exemple, « IEEE std 829-1983 ; standard for preparing a system test plan ».

Dans le document, il n'y a pas d'explications supplémentaires ni d'exemples. Nous ne savons donc pas ce que la rubrique « Security Exceptions » signifie exactement. Le document

---

[12] L'auteur spécifie que les concepts ont été empruntés à Rumbaugh ([Rumbaugh 94]) et personnalisés par WorldCom.

se rapporte à des « use case » qui sont, comme on l'a dit à la section 3.2.2, des cas particuliers de scénarios.

### C. Regnell

Regnell décrit dans son document [Regnell 96] un cadre conceptuel des « use case » dans lequel il fait intervenir la notion de service et celle de but (cfr. Figure 28).

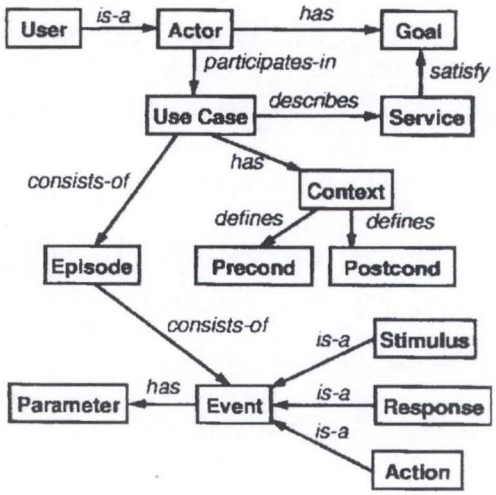



**Figure 28 - cadre conceptuel des « use case » selon Regnell**

Un « use case » (ou scénario) décrit un service qui satisfait un certain but. Le but est définit par l'acteur intervenant dans le scénario. Cette vue nous pousse à considérer le but des acteurs intervenant dans les scénarios (et pas le but du scénario) ainsi que les services que décrit le scénario comme des rubriques intéressantes.

### D. Crews

Dans un but de classification des scénarios selon leur contenu, les vues du contenu décrivent dans [CREWSD1-I1.1] peuvent être d'une grande aide. Nous pensons à des rubriques correspondant aux facettes du « content view », à savoir :

- le contexte (que nous appellerons la portée pour éviter toute confusion) : indique si le scénario décrit le comportement interne du système, l'interaction entre le système et son environnement, le contexte organisationnel, etc. ;

- l'argumentation : indique si le scénario est utilisé pour discuter des positions alternatives par rapport à un problème, pour confirmer ou infirmer une certaine position, pour décrire des problèmes ou conflits et/ou pour pouvoir choisir une certaine position ;

Le niveau d'abstraction et la couverture (fonctionnel, non-fonctionnel ou intentionnel – cfr. 3.5.4) sont des facettes d'un intérêt moins important pour les scénarios ALBERT.

Dans plusieurs documents et livres, les auteurs ajoutent des dessins de la situation dans laquelle se déroule le scénario. Il peut s'agir d'un dessin d'un interface homme-machine ou simplement de la description de l'environnement. Dans le cadre du problème du carrefour, un dessin de celui-ci peut aider fortement à la compréhension du scénario. On peut donc considérer un dessin du contexte comme rubrique supplémentaire.

## 4.3 Les scénarios ALBERT

Le but de cette section est de définir un langage d'expression de scénarios particulièrement adapté au langage ALBERT.

Après un rappel des points importants à prendre en considération lors de la construction d'un tel langage (section 4.3.1), nous allons identifier quelques particularités des scénarios ALBERT (section 4.3.2). Ensuite, nous définirons un langage pour le contenu (section 4.3.3) et puis un langage pour le contexte (section 4.3.4) des scénarios ALBERT. Nous terminerons par un exemple complet (section 4.3.6).

### 4.3.1 Rappel des points importants

L'élément le plus important est que le langage utilisé pour exprimer les scénarios est le mieux adapté possible à la situation effective. Il faut savoir qui va les manipuler et dans quel but. Le langage doit d'une part permettre d'exprimer tout ce qui est important et d'autre part soutenir celui qui l'utilise en lui fournissant des structures de base et des manipulations aisées sur ces structures.

Les besoins de l'utilisateur de scénarios peuvent être traduits en termes de représentation pouvant varier suivant les 5 dimensions détaillées à la section 3.4.2. Il faut donc fixer le ou les médiums utilisés, le niveau de formalité, le niveau d'interactivité, le niveau de détail et la structuration entre scénarios. Un certain utilisateur de scénarios dans un certain but a besoin d'une vue particulière.

En ce qui concerne le contenu, le langage joue un rôle un peu moins important. Il peut bien sûr limiter le niveau d'abstraction et proposer des repères pour toutes les informations intéressantes en rapport avec les scénarios (auteur, but, date de création, etc.). Cependant l'élément important est que sa forme permette d'exprimer le contenu souhaité.

### 4.3.2 Les particularités des scénarios ALBERT

Notre souhait est d'arriver à un langage d'expression de scénarios permettant de tirer des conclusions sur une spécification écrite en ALBERT. Ces conclusions peuvent être obtenues directement à partir du scénario ou bien à travers une animation de la spécification basée sur des scénarios. Dans tous les cas, la correspondance entre le langage ALBERT et les scénarios ALBERT doit être relativement immédiate.

Nous savons également que le langage sera essentiellement utilisé par des analystes. Cependant, il peut servir de moyen de communication entre l'analyste et le client ou

l'utilisateur final. Cela est surtout vrai lorsque le scénario exprime le résultat d'une animation, par exemple.

Ces remarques nous permettent de réduire un peu le champ des formes d'expressions possibles.

Nous savons déjà que la langage ALBERT permet de définir un vocabulaire servant de base pour les contraintes formelles. Le langage de scénarios doit se baser sur le même vocabulaire et donc supporter les mêmes constructions de base, à savoir les agents, les actions et les composants d'état.

D'un autre côté, la notion de message est inexistante en ALBERT et il serait donc plus facile d'avoir un langage de scénarios ne faisant pas intervenir ce concept. En effet, la communication entre des agents en ALBERT se fait grâce à des contraintes de mise à disposition d'une information et à des contraintes de perception de cette information. Dans le cadre des action, on peut trouver un rapprochement à la notion habituelle de message : le début et la fin de la perception d'une occurrence d'une action peut être considéré comme une réception d'un message. Dans le cadre des composants d'état, la perception peut être est continue. Modéliser cela par un échange de message discret n'est dès lors pas à conseiller.

Le vocabulaire défini en ALBERT permet de fixer le niveau de détail des scénarios. Comme nous l'avons dit plus haut, cela permet d'éviter un problème important : la détermination du niveau de détail.

Nous avons aussi besoin d'un langage assez formel pour pouvoir faire facilement une correspondance entre le langage de scénario ALBERT et une spécification ALBERT. Ainsi, certains traitements peuvent être automatisés. Cependant, il ne faut pas oublier que le langage peut servir pour la communication entre l'analyste et le client. Il doit donc être facilement lisible. Parmi les langages formels, un langage avec support graphique sera donc préférable.

Les scénarios peuvent servir de base pour faire une animation de la spécification mais ils ne sont pas l'animation. Leur niveau d'interactivité n'est donc pas très élevé.

Les besoins au niveau de la structuration sont assez généraux : il faut pouvoir retrouver facilement un scénario parmi plusieurs autres, il faut pouvoir créer des liens entre plusieurs scénarios et indiquer le type de ce lien.

Tout cela restreint le choix du langage d'expression des scénarios et nous permet donc de nous focaliser sur les éléments importants.

### 4.3.3 Le langage du contenu

#### A. Introduction

Parmi les langages examinés dans la section précédent, les MSC semblent être le meilleur moyen d'exprimer des scénarios. Cela est surtout dû aux extensions proposées par différents auteurs. En effet, les diagrammes MSC de base sont moins intéressants que les diagrammes de flux, par exemple, parce que ces derniers représentent en plus les actions effectuées par les

différents acteurs lors de la réception d'un message. C'est donc uniquement grâce aux extensions que les MSC deviennent plus intéressants.

Avant de rentrer dans le vif du sujet, nous allons donc parcourir les différentes extensions et nous allons discuter leur utilité. Ensuite, nous montrerons comment les MSC peuvent exprimer des scénarios ALBERT.

## B. Les extensions des MSC

A travers la littérature, nous trouvons plusieurs extensions intéressantes : l'introduction de boucles, points de décisions, exécutions parallèles, exécutions exceptionnelles, structuration des MSC, représentation des actions, etc. Cette section a pour but d'expliquer toutes ces extensions.

Les MSC de base sont conçus pour exprimer des scénarios fortement instanciés. A cause de cela, les boucles ne sont pas exprimables. En effet, lorsqu'on sait qu'un certain passage est parcouru 4 fois, on l'écrit 4 fois l'un à la suite de l'autre. Le scénario est donc exprimable sous forme d'un MSC. Lorsque la répétition dépend de la réalisation d'une condition ou doit se faire n fois, il est plus utile d'exprimer cela sous la forme d'une boucle. Les MSC de base ne permettent pas d'exprimer ces scénarios.

Nous allons discuter les extensions proposées par deux auteurs différents. D'une part l'ITU-T, qui dans sa dernière version du standard MSC distingue les MSC de base et les MSC de haut niveau, et d'autre part Björn Regnell [Regnell 96], qui propose une représentation des répétitions directement sur les MSC de base.

« Le standard MSC, comme il est défini par la recommandation Z.120 [12][13] de l'ITU-T, introduit deux concepts de base : MSC de base (bMSC) et MSC de haut niveau (hMSC). Un bMSC consiste en un ensemble de processus qui s'exécutent en parallèle et échangent des messages de manière asynchrone. Un hMSC combine graphiquement des références vers des bMSC pour décrire une exécution parallèle, séquentielle, itérative et non-déterministe de bMSC. » [Abdallah 97]

Nous avons déjà donné un exemple de bMSC dans la section 4.2.1 (Figure 23). La Figure 29 représente un exemple de hMSC.

---

[13] [12] ITU-T, Recommendation Z.120, ITU – Telecommunication Standardization Sector, Geneva, Switzerland, May 1996. Review Draft Version.

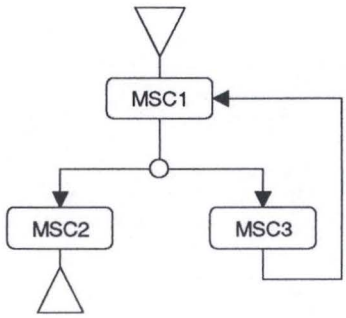



**Figure 29 - exemple de hMSC**

Les différentes combinaisons possibles sont :

- l'exécution parallèle AND (la fin d'un bMSC déclenche deux bMSC),
- l'exécution séquentielle (la fin d'un bMSC déclenche un bMSC),
- l'itération (il existe une boucle dans le graphe des bMSC),
- l'exécution non-déterministe OR (cfr. l'exemple Figure 29 - exemple de hMSC).

Il n'y a pas de support pour la modélisation de situations d'exception pouvant interrompre un bMSC. Il n'y pas non plus de support explicite pour exprimer des conditions lors d'un branchement OU (du genre si ... alors ... sinon).

L'avantage principal des extensions décrites ci-dessus est bien sûr le gain évident en expressivité et en flexibilité lors de la description de scénarios. A part cela, elles permettent d'approcher les scénarios en deux étapes, d'un part l'enchaînement des grandes étapes ou tâches principales et d'autre part la coopération entre acteurs dans le but de réaliser ces grandes étapes. Ces extensions apportent donc aussi un moyen de structuration des scénarios.

Dans leur thèse [Andersson 95] Michael Andersson et Johan Bergstrand essayent d'adapter le formalisme des MSC pour exprimer le contenu des « use case ». Pour cela, ils définissent 3 niveaux de détail des « use case » ainsi qu'un langage associé à chaque niveau. Les trois niveaux s'appellent le niveau système (system level), le niveau structurel (structure level) et le niveau de base (basic level).

Le niveau structurel est comparable aux hMSC de l'ITU-T. Il combine des MSC de base pour obtenir des exécutions séquentielles, des alternatives, des répétitions, des exceptions et des interruptions. En plus, il existe un mécanisme pour maîtriser la complexité : on peut grouper la combinaison de plusieurs blocs (soit bMSC, soit hMSC) en un seul nouveau bloc. Ce nouveau bloc peut alors être manipulé comme une seule entité.

Le niveau de base possède les mêmes mécanismes mais exprimés avec un niveau de détail plus élevé. Un opérateur du type répétition, alternative et exception est représenté par des rectangles entourant la partie à répéter, l'alternative ou l'exception. L'interruption est signalée par un petit rectangle à côté identifiant le MSC s'exécutant lors d'une interruption. Il y a

moyen de combiner des opérateurs. Les deux figures suivantes (Figure 30 et Figure 31) montrent les représentations associées à ces concepts.

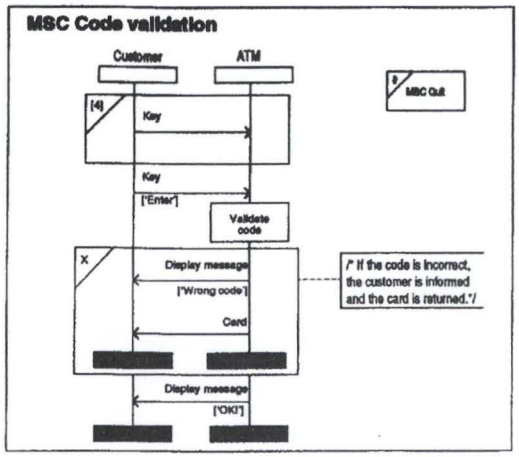



**Figure 30 - exemple de répétition, exception et interruption [Andersson 95]**

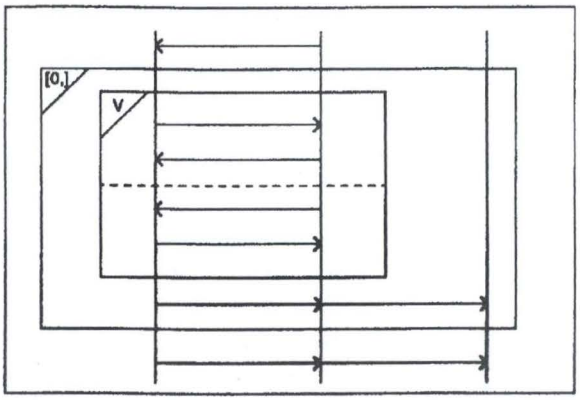


**Figure 31 - exemple d'encapsulation d'opérateurs [Andersson 95]**

Un autre mécanisme inexistant au niveau des MSC initiaux est la représentation de la création et de la destruction du processus. Dans le cadre d'un design orienté objet, il est très important de pouvoir représenter cette construction. Cependant au niveau de l'ingénierie des besoins, l'utilité reste assez limitée.

Björn Regnell parle d'une structuration des scénarios en épisodes [Regnell 96]. Un épisode correspond à la construction de base au niveau structurel. A la Figure 30, 'MSC Code validation', 'Validate Code' et 'MSC Quit' sont des épisodes. Les bMSC de l'ITU-T constituent également des épisodes. Cette structuration facilite la réutilisation de certaines parties de scénarios à l'intérieur d'autres scénarios. A côté de la structuration importante intra-scénarios, on peut donc obtenir une structuration inter-scénarios assez importante.

L'avantage de ces notations sur celles proposées par l'ITU-T est bien sûr qu'elles supportent d'une part des constructions de type interruption et extension et d'autre part qu'il existe un mécanisme de structuration important inexistant dans la notation de l'ITU-T.

Le problème rencontré surtout lorsqu'il s'agit de représenter un scénario assez compliqué au niveau 'de base' est la surcharge du dessin. On a un aperçu à la Figure 31 où l'imbrication de deux opérateurs rend le schéma presque illisible.

Un autre problème lié au fait que des répétitions sont représentées directement sur le MSC de base est que l'axe du temps n'a plus de sens. Une caractéristique importante des MSC de base est que le temps est strictement croissant de haut en bas. Dans l'exemple ci-dessus, ce n'est plus le cas. Toutes les vérifications éventuelles concernant des propriétés temps-réel sont beaucoup plus difficiles.

Une extension des MSC rencontré chez Rumbaugh, Booch et Jacobson [UML] est la représentation explicite du contrôle. La ligne du temps associée à un acteur est représentée de manière plus épaisse à l'endroit où l'acteur a le contrôle. La Figure 32 nous montre un exemple.

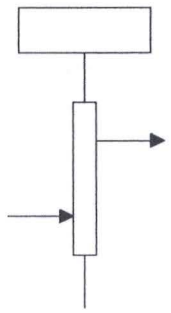


**Figure 32 - exemple de représentation du contrôle**

Ce mécanisme peut être utilisé pour indiquer la durée d'une action sur l'axe du temps d'un acteur. Il suffit d'associer une action à chaque 'élargissement' de la ligne du temps.

L'ITU-T a également défini des extensions permettant d'exprimer des contraintes temporelles. « Il spécifie essentiellement 4 classes de constructions permettant d'exprimer des contraintes temporelles : timers, intervalles de délai, règles de dessin et autres marquages temporels.

Timers expriment des contraintes temporelles dans le bMSC. Dans un même processus, un timer peut être initialisé à une certaine valeur, mis à zéro et on peut observer un timeout. » [Abdallah 97].

Les intervalles de délai fixent un temps minimal et maximal qui peut s'écouler entre l'occurrence de deux événements. Un événement est l'envoi ou l'arrivée d'un message.

Les deux dernières constructions, à savoir les règles de dessin et les autres marquages temporels viennent des extensions apportées par UML. La plus importante est le marquage du contrôle sur la ligne du temps.

D'autres extensions proposées concernent des concepts orientés objet applicables aux MSC. Ils incluent entre autre l'héritage, la virtualité et la redéfinition. Nous ne disposons pas de renseignements sur leur implémentation à l'heure actuelle.

## C. Les scénarios ALBERT - aperçu

Nous avons essayé d'exploiter la richesse et la précision des MSC, et surtout des dernières extensions de ceux-ci, pour représenter des scénarios ALBERT. Dans cette section, nous expliquons comment les MSC peuvent être utilisés pour exprimer des scénarios ALBERT.

Les concepts de base du langage ALBERT sont les agents, les actions, les états et les contraintes. Dans le cadre des scénarios, nous portons un intérêt particulier aux agents et les instances de leurs actions au cours du temps.

On peut faire facilement la correspondance entre les agents du langage ALBERT et les acteurs ou processus des MSC. Ils sont dans tous les cas au niveau instance même si l'instance est indéterminée (la voiture X au lieu de la voiture de monsieur Dupond).

Par contre au niveau des actions, on rencontre certaines difficultés : les MSC savent exprimer la durée d'une action grâce à la ligne du temps 'élargie' mais les actions ALBERT peuvent avoir lieu en parallèle, même pour un même agent.

Nous devons donc introduire une extension supplémentaire pour pouvoir représenter cette situation. Lorsqu'une action est déclenchée alors qu'une autre est encore en cours, une déviation par rapport à la ligne de temps est créée. Cette déviation rejoint la ligne originale à la fin de l'action. La Figure 33 donne un exemple de représentation d'actions s'exécutant en parallèle.

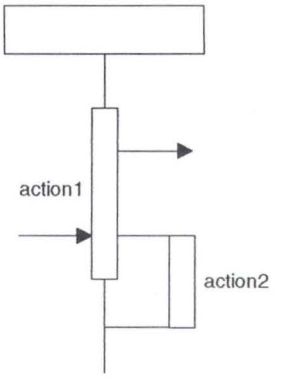



**Figure 33 - exemple d'actions s'exécutant en parallèle**

Il est bien sûr important de pouvoir exprimer des scénarios comportant des boucles, des alternatives, des exceptions, des interruptions et des déroulements en parallèle. Vu que des extensions aux MSC existent pour exprimer cela, nous les adoptons.

Nous allons prendre des deux approches relatives aux extensions de ce type, les éléments le plus adaptés pour notre problème. De l'ITU-T, nous adoptons la distinction entre MSC de base (bMSC) et MSC de haut niveau (hMSC). Les opérateurs liant différents bMSC entre eux dans un hMSC sont les suivants :

- composition séquentielle,

- composition parallèle,
- composition alternative (avec la possibilité de fixer des conditions aux branchements),
- indication de cheminement exceptionnel (comportement anormal), et
- interruption.

Des boucles sont alors obtenues grâce à une combinaison de compositions séquentielles et alternatives. Nous ne reprenons pas les extensions de création et de destruction de processus. En effet, ALBERT n'admettant pas des contraintes de création ou destruction d'agents, il est tout a fait inintéressant de les prévoir dans le langage des scénarios ALBERT.

De Regnell et al. [Andersson 95], [Regnell 96] nous empruntons le mécanisme de structuration. Un hMSC peut être constitué de un ou de plusieurs bMSC et de zéro ou plusieurs hMSC. Un hMSC sera appelé un épisode. L'épisode joue un rôle important dans la structuration des scénarios.

Grâce aux travaux de Hanêne Ben-Abdallah et Stefan Leue [Abdallah 97], nous obtenons des indications précieuses sur le traitement des contraintes temporelles exprimées dans les scénarios. De manière général, on distingue l'analyse du temps au niveau des bMSC et l'analyse du temps au niveau des hMSC. Au niveau des bMSC, tout est séquentiel du haut vers le bas. Un événement représenté plus haut qu'un autre signifie qu'il a lieu avant l'autre. A niveau des hMSC, il suffit de reporter le temps de fin d'un MSC comme temps de début du suivant.

Il nous reste à faire la correspondance entre les messages des MSC et les 'messages' d'ALBERT. Le but étant de pouvoir tirer des conclusions sur une spécification écrite en ALBERT à partir d'un scénario écrit dans le formalisme des MSC, il est nécessaire de faire cette correspondance.

Comme déjà remarqué plusieurs fois, le concept de message est inexistant dans ALBERT. La seule situation correspondant vaguement à un échange de message est celle ou un agent perçoit une action d'un autre agent et réagit en fonction de cette perception. Nous pouvons aussi dire que la perception d'un composant d'état constitue une sorte d'échange en continu de messages entre deux agents.

Ce qui est clair, c'est que l'échange d'information entre agents se fait grâce aux contraintes de coopération (information d'état, perception d'état, information d'action et perception d'action). Nous allons redéfinir la sémantique des flèches entre les processus (agents dans le cas des scénarios ALBERT) d'un bMSC en fonction de ces concepts.

Une flèche représentera le fait qu'une certaine situation chez l'agent d'origine a un effet sur la situation chez l'agent de destination. L'effet est dans tous les cas le déclenchement d'une action. Pour qu'il y ait effet chez l'agent de destination, il faut qu'il ait perçu quelque chose. Ce qui a été perçu sera indiqué sur la flèche. Il peut s'agir de l'occurrence d'une action ou d'une certaine valeur d'un composant d'état.

Vu que l'effet peut avoir lieu longtemps après que la condition ait eu lieu, les flèches peuvent être obliques. L'origine de la flèche se rapporte au moment ou la condition de déclenchement est réalisée et la pointe de la flèche indique le moment du début de l'effet.

La Figure 34 donne un exemple de cette nouvelle sémantique : le début de l'action *entre_dans_carrefour* de l'agent *voiture* a été déclenchée par la condition *couleur = vert depuis une seconde* chez l'agent *feu*. Le déclenchement est immédiat après la perception de la couleur pendant une seconde.

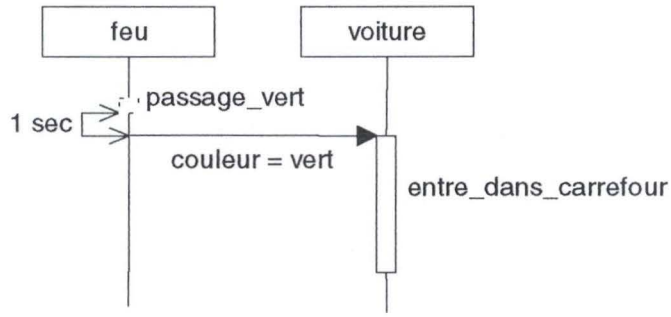



**Figure 34 - exemple de nouvelle sémantique des flèches**

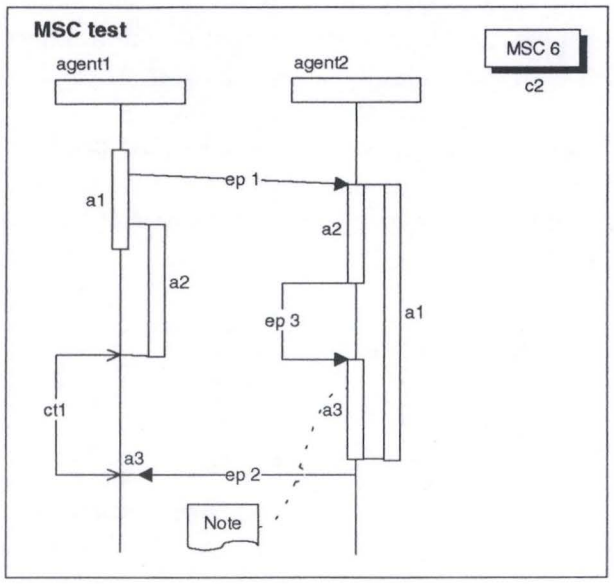
On peut très bien avoir des flèches partant d'un agent pour aller vers ce même agent. Cela correspond alors à une contrainte de type 'triggering' dans une spécification ALBERT.

Cette section a donné un aperçu de la manière dont on peut exploiter les MSC munis de leurs extensions pour représenter des scénarios ALBERT. Dans la section suivante, nous allons rappeler la syntaxe des MSC adaptés aux scénarios ALBERT. Ensuite nous définirons de manière un peu plus précise la sémantique associée à cette notation.

## D. Les scénarios ALBERT– syntaxe

Rappelons que les MSC servent uniquement à exprimer l'aspect fonctionnel. Ils permettent de représenter le contenu et non le contexte des scénarios.

Rappelons aussi la distinction entre les bMSC et les hMSC. Les premiers expriment des épisodes de base et les dernier permettent de combiner ceux-ci entre eux. Nous commençons par la syntaxe de bMSC. La Figure 35 représente les éléments essentiels d'un bMSC.




Légende :
- a1, a2, a3 sont des actions
- ep1, ep2 et ep3 sont des éléments perçus
- ct1 est une contrainte temporelle

**Figure 35 - éléments de construction de bMSC**

Chaque bMSC est représenté par un cadre à l'intérieur duquel on trouve son nom considéré comme identifiant. En dessous de son nom sont alignés un certain nombre de rectangles représentant les agents intervenant dans l'épisode représenté. De chacun de ces rectangles part une ligne verticale vers le bas représentant la ligne du temps. Les rectangles sont munis d'un texte représentant le nom de l'agent.

La ligne du temps d'un agent peut s'élargir sur un certain morceau (a1, a2). Cet élargissement est muni d'un libellé indiquant une action. Une barre horizontale coupant la ligne du temps représente une action instantanée (a3). Elle représente un élargissement sur une longueur zéro. On peut dédoubler une ligne du temps sur n'importe qu'elle longueur (a2).

On peut dessiner des flèches partant d'une ligne du temps vers le début d'un élargissement d'une autre ligne du temps ou de la même ligne du temps. La seule contrainte à respecter est que le point d'origine de la flèche soit plus haut ou à même hauteur que la pointe de la flèche. Une flèche est libellée (ep1, ep2).

On peut également définir un ou plusieurs rectangles ombrés (MSC 6). Ils représentent les épisodes d'interruption et sont munis d'un texte décrivant la condition d'interruption (c2). La manière d'écrire cette condition sera explicitée plus tard.

On peut également dessiner des flèches avec deux pointes arrivant et partant d'une ligne du temps. Ces flèches sont libellées et représentent un intervalle de temps. Le texte associé est une contrainte temporelle dont la syntaxe est la suivante : [a, b], avec a et b des variables et/ou des chiffres.

Des notes informelles peuvent être attachées à n'importe quel élément du graphique grâce au symbole réservé à cet effet (note) et à une ligne pointillée. Cette règle est aussi applicable aux hMSC.

La syntaxe des hMSC se base sur un autre ensemble de symboles représentés à la Figure 36.

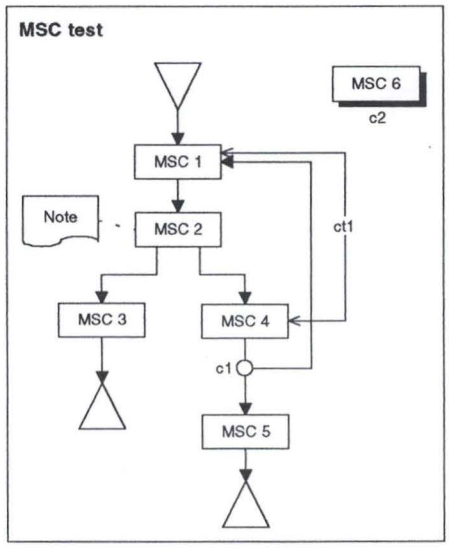



**Figure 36 - les éléments de base pour les hMSC**

Chaque hMSC est représenté par un rectangle dans lequel est inscrit le nom en haut à gauche, identifiant l'épisode représenté.

On trouve exactement un rectangle 'inversé' et un ou plusieurs rectangles 'droits' sur un hMSC. Le premier représente le point de départ et le dernier le point de sortie de l'épisode.

D'autres rectangles contenant des noms d'autres épisodes sont représentés à l'intérieur du rectangle principal. Ils représentent les épisodes à combiner pour obtenir l'épisode représenté. La combinaison se réalise en dessinant des flèches entre les rectangles. Par rectangle, on a une ou plusieurs flèches sortantes et une ou plusieurs flèches entrantes. Les flèches vont du haut vers le bas, sauf pour indiquer une boucle.

On peut également définir un ou plusieurs rectangles ombrés. Ils représentent les épisodes d'interruption et sont munis d'un texte décrivant la condition d'interruption (c2). La manière d'écrire cette condition sera explicité plus tard.

Une autre construction syntaxique est un petit rond. Il peut être lié avec une ligne à un ou plusieurs rectangles et donne lieu à deux ou à plusieurs flèches allant vers des rectangles. Le rond est muni d'un libellé dont la syntaxe est la même que pour les conditions d'interruption (c2).

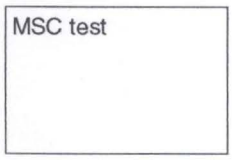
Tout comme pour les bMSC, des flèches à deux pointes peuvent être utilisées pour exprimer des contraintes temporelles. La syntaxe reste la même.

Lors d'un support de cette notation par un outil case, on peut prévoir des facilités de navigation en permettant à un utilisateur de voir les détails d'un épisode représenté uniquement par un rectangle. L'utilisateur peut ainsi 'zoomer' jusqu'au bMSC et revenir au hMSC de plus haut niveau. Dans le cas où aucun support informatique existe, il est conseillé de noter chaque épisode sur une feuille à part et d'indexer ces feuilles pour pouvoir retrouver facilement l'information voulue. Utiliser une seule feuille par épisode a aussi l'avantage de pouvoir noter les informations contextuelles d'un côté et les informations sur le contenu de l'autre.

## E. Les scénarios ALBERT – sémantique

Dans cette section, nous allons donner une sémantique de base aux différents éléments intervenants dans la notation MSC. On utilisera la sémantique habituelle des MSC pour tous les éléments non spécifiques aux scénarios ALBERT. Les différences entre les MSC standards et les MSC adaptés aux besoins des spécifications ALBERT se remarquent surtout ici.

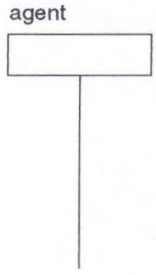
Nous allons commencer par les bMSC et passer en revue toutes les constructions syntaxiques pour leur associer une sémantique.




Un rectangle contenant un texte en haut à gauche représente un épisode. Un épisode est une partie d'un scénario ou même un scénario tout entier.

Le rectangle peut contenir d'autres informations relatives à l'épisode et il peut être vide. Dans ce cas, il représente une référence vers l'épisode identifié par le nom en haut à gauche.

La construction à droite représente une partie d'une vie d'un agent ALBERT. L'agent représenté est identifié par le nom se trouvant au-dessus de la partie rectangulaire du dessin.

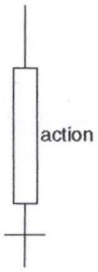



Il s'agit d'une partie d'une vie ALBERT parque que une vie ALBERT n'a pas de début, ni de fin et que la représentation ici à droite a un début et une fin.

Le temps 'avance' de haut en bas. L'avancement n'est pas nécessairement continu, c'est-à-dire que deux intervalles de même longueur sur la ligne verticale ne représentent pas nécessairement la même durée. Le temps représenté entre deux points peut être indiqué par des contraintes temporelles (cfr. plus loin).

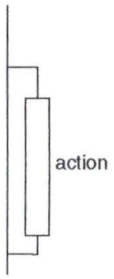
Deux ou plusieurs agents peuvent être représentés l'un à côté de l'autre. Dans ce cas, les rectangles au-dessus de la ligne du temps doivent se trouver à la même hauteur. Deux points sur la ligne du temps de deux agents se trouvant à une même distance par rapport à ce rectangle indiquent le même moment.

Dans la suite, nous nommeront la ligne verticale la ligne du temps de l'agent.




Sur le dessin à gauche, la ligne du temps est élargie sur un certain intervalle. L'élargissement est muni d'un texte. Il représente l'occurrence d'une action dont le nom est donné par le texte.

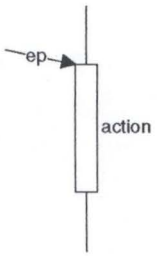
La longueur de l'intervalle donne une indication sur la durée de l'action. Un élargissement le longueur 0, c'est-à-dire une ligne horizontale (comme on la voit en bas du dessin), représente une action instantanée.




La déviation vers la droite du dessin représentée à droite est une déviation de la ligne du temps. Une déviation d'une ligne du temps reste une ligne du temps associée au même agent. La sémantique est donc la même.

L'intérêt d'une déviation est de pouvoir représenter des occurrences d'actions d'un même agent ayant lieu au même moment.

On peut créer autant de déviations qu'on veut pour un même agent.

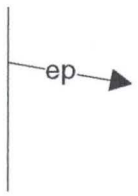



La flèche arrivant au début de la représentation d'une occurrence d'une action (cfr. dessin à gauche) représente le fait que l'occurrence de cette action est due à un élément perçu d'un autre agent ou à une condition interne.

L'élément perçu ou la condition interne est identifié par le texte associé à la flèche.

Un élément perçu peut être l'occurrence d'une action ou l'état d'un composant d'état d'un autre agent. Une condition interne peut aussi être l'occurrence d'une action ou l'état d'un composant d'état.

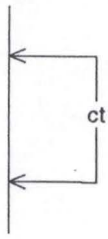
Un flèche partant d'une ligne du temps (cfr. dessin à droite) indique le moment de perception d'un élément d'un autre agent ou d'une condition interne.




Lorsque l'élément perçu est la valeur d'un composant d'état, alors le moment indiqué par l'origine de la flèche signifie le moment où la condition déclenchant un effet chez un autre agent est rempli pour la première fois. Par exemple, si la voiture démarre (effet) après qu'elle ait vu pendant une seconde le feu vert (condition), alors l'origine de la pointe sera à une seconde après le passage au vert du feu. Dans le cas d'une perception d'un composant d'état, on trouve souvent une contrainte temporelle juste avant le flèche (pour indiquer une seconde, par exemple).

Lorsque l'élément perçu est l'occurrence d'une action, alors l'origine de la pointe est placée sur le point indiquant le début de l'action perçue.

Une flèche oblique indique que l'effet n'a lieu qu'un certain temps après la perception. Une flèche horizontale signifie donc que l'effet a lieu immédiatement après.




Une flèche à deux pointes donne une contrainte temporelle entre deux points sur la ligne de temps. Cette flèche peut être dessinée directement sur les lignes du temps d'un ou de plusieurs agents ou sur une ligne du temps parallèle aux autres lignes du temps destinées uniquement à représenter les contraintes temporelles. Grâce à la règle que deux points à la même hauteur représentent des mêmes moments, cela ne pose aucun problème.

Une contrainte de temps à la forme [a, b]. Elle signifie que le temps que représente l'intervalle indiqué par la flèche est compris entre a et b. Ainsi, si une flèche à deux pointes est dessinée entre le début et la fin d'une représentation d'un occurrence d'une action, et si la contrainte temporelle est [2 sec, 4 sec], alors cela signifie que cette occurrence d'action dure entre 2 et 4 secondes.

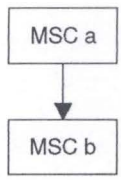
Le dessin à droite n'a aucune importance sémantique. Il sert uniquement à pouvoir attacher des notes à des éléments du graphe.


Les rectangles ombrés représentent des épisodes d'interruption. Ils sont munis d'un texte représentant la condition sous laquelle une interruption a lieu. La condition est écrite dans le même formalisme que les contraintes de type 'state behavior' et se rapportant au méta-modèle défini avec le langage de contexte formel (cfr. plus loin, section C).

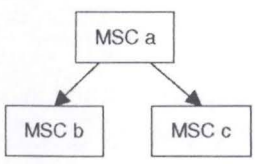
Après l'explication de la sémantique des bMSC, nous allons détailler celle associée aux hMSC qui est assez intuitive. La sémantique des hMSC consiste essentiellement à définir les temps de débuts et le temps de fin des épisodes intervenant dans la construction du hMSC.

Les constructions de base sont les rectangles à l'intérieur desquels on trouve le nom d'un MSC (de base ou de haut niveau). Un tel rectangle représente donc un épisode. Sa sémantique est celle associée à sa représentation détaillée de bMSC ou de hMSC.

Les rectangles sont interconnectés par des flèches. La signification des différentes interconnexions possibles sera détaillée maintenant.



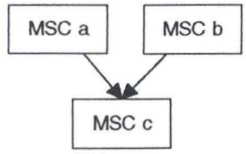

Une flèche partant d'un rectangle A et arrivant à un autre rectangle B (cfr. dessin à droite) représente l'enchaînement séquentiel des épisodes. Cela signifie que le temps de début de l'épisode B est égal au temps de fin de l'épisode B.




Si deux ou plusieurs flèches partent d'un même rectangle vers deux ou plusieurs rectangles différents, cela signifie que les épisodes pointés commencent tous au même moment. Ce moment est le temps de terminaison de l'épisode origine.

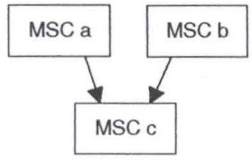
Cette construction est nommée l'enchaînement parallèle ou l'enchaînement AND. Le dessin à gauche en donne un aperçu.

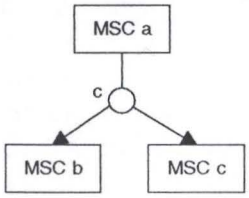
Lorsque deux ou plusieurs flèches arrivent à un même rectangle, deux situations se présentent : ou bien elles le rejoignent au même endroit ce qui indique une synchronisation, ou bien elles le rejoignent à un endroit différent, ce qui signifie une duplication de l'épisode.



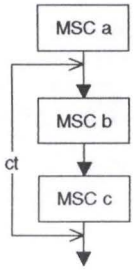

Dans le cas d'une synchronisation, le temps de début du MSC pointé est égal au temps de fin du dernier MSC d'origine.

Dans le cas de la duplication est souvent lié aux boucles. Lorsqu'on parcourt le hMSC décrit et lorsqu'on passe par une des flèches en question, le temps de début est le même que le temps de fin du dernier épisode parcouru. Sur le dessin à droite, si on vient du MSC a, le temps de début du MSC c est le temps de fin du MSC a. Si on vient du MSC b, le temps de début du MSC c est le temps de fin du MSC b.




Un petit rond muni d'un libellé indique un point de décision. Le libellé représente la condition qui est écrite et interprétée de la même manière que les conditions des MSC d'interruption. En fonction du résultat de l'évaluation de la condition au temps actuel (le temps de fin du dernier MSC parcouru), un des scénarios pointé sera parcouru. Son temps de début sera le temps de fin du dernier MSC parcouru. La condition peut faire référence à des variables définies au niveau du schéma et non au niveau de la spécification ALBERT liée au scénario. Une tel variable peut être intéressante pour comptes le nombre de boucles, par exemple. Nous appellerons une telle variable 'méta-variable'.




Le triangle droit représente une sortie du scénario et le triangle inverse représente le point d'entré du scénario.

Tout comme pour les bMSC, les flèches à deux pointes peuvent être utilisées pour exprimer des contraintes temporelles. Les rectangles ombrés ont également la même représentation et la même sémantique que celles rencontrés pour les bMSC.

### 4.3.4 Le langage du contexte

#### A. Introduction

Comme déjà expliqué plus haut, nous distinguons deux aspects différents pour le contexte d'un scénario. Les informations se rapportant d'une part à la création et la gestion des scénarios et d'autre part à la mise en situation intéressante par rapport au scénario.

Nous définissons deux langages, un pour chaque aspect. Le premier est plutôt informel et se base sur les différentes rubriques rencontrées dans la littérature (cfr. état de l'art section 4.2.2). Le non-formel a été choisi parce qu'il permet d'atteindre la flexibilité nécessaire dans l'expression de ce genre d'information.

Le deuxième est formel et se base sur des idées empruntées au langage ALBERT. Il est d'une grande importance dans le cadre d'une animation d'une spécification ALBERT et le choix d'un langage formel est essentiellement motivé par cela. On peut ajouter à cette raison le fait que la mise en situation intéressante d'un scénario utilise souvent les mêmes constructions de phrases et se base sur un vocabulaire utilisé dans la spécification ALBERT correspondante.

Dans une première partie, nous allons définir le langage informel et dans une deuxième le langage formel utilisé pour décrire le contexte.

## B. Le contexte de création du scénario

### i) Les adaptations aux scénarios ALBERT

Nous allons grouper les rubriques rencontrées dans la littérature en unités logiques et les adapter aux besoins des scénarios ALBERT.

Certaines rubriques examinées lors de l'état de l'art ne sont pas utiles dans la partie informelle du langage du contexte. Ainsi, le « interaction diagram » et le « flow of events » de [SEED 95] sont couverts par le langage du contenu détaillé ci-dessus (C). De même, les préconditions et postconditions font parties du langage formel du contexte.

La rubrique « associations » [SEED 95] ou « related use cases » [Vemulapalli] n'a d'importance que s'il existe une structuration inter-scénarios. Dans notre cas, cette structuration est obtenue grâce aux mécanismes l'encapsulation au niveau des hMSC.

Les « special requirements » [SEED 95] ne sont pas nécessaires non plus. Ils font partie de la spécification ALBERT associée au scénario.

Les rubriques restantes peuvent être regroupées comme suit.

- *Les rubriques de spécification du scénario.* On y classe le nom du scénario, sa brève description, son but (y compris la facette de l'argumentation [CREWSD1-I1.1]), le service qu'il décrit et les agents y intervenant. On peut en plus ajouter une rubrique reprenant des mots clés.

- *Les rubriques des informations de création du scénario.* On y trouve les noms et coordonnées des personnes ayant fourni des informations pour la création des scénarios, les noms et coordonnées des personnes ayant écrit le scénario, la date de création, la version et les traces permettant de retrouver les débuts de l'élaboration du scénario.

- *Les rubriques de documentation supplémentaire.* On y trouve éventuellement un ou des dessins expliquant le contexte ou l'environnement, les références bibliographiques ayant aidé à élaborer le scénario, la spécification ALBERT fournissant le vocabulaire et les notes libres.

Certaines des rubriques sont obligatoires, d'autres ne le sont pas. En fonction de la portée du scénario ou de l'épisode, certaines rubriques sont plus intéressantes que d'autres. Le

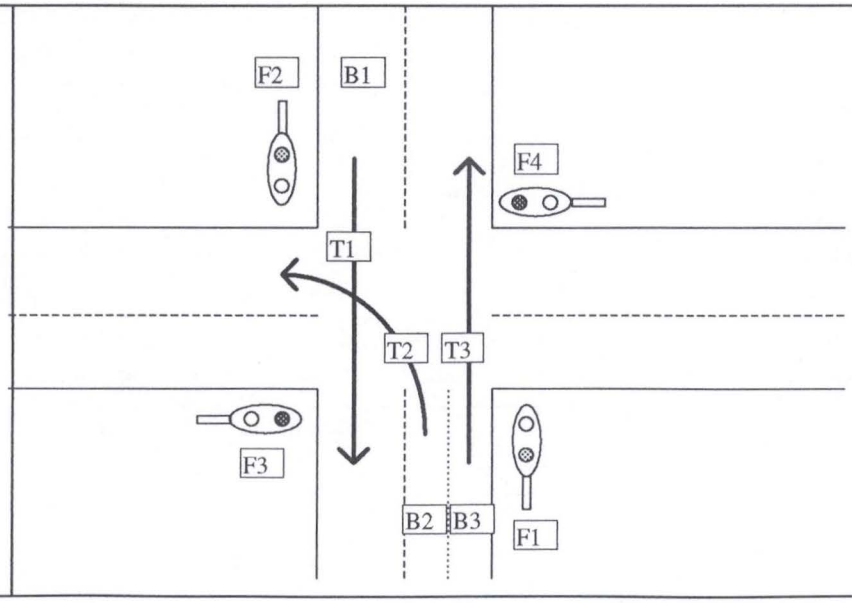
service rendu est, par exemple, plus intéressant pour un 'grand' scénario que pour un épisode décrit par un seul bMSC.

Le tableau ci dessous résume le modèle complet que nous avons défini.

| **Nom rubrique** | **Description** |
|---|---|
| *Spécification du scénario* | |
| Nom | Identifiant de l'épisode ou du scénario. |
| Brève description | Deux ou trois lignes décrivant de manière informelle le scénario. |
| But | Objectif du scénario. Cela comporte d'une part le pourquoi de ce scénario et d'autre part les objectifs de certains agents intervenant dans le scénario. |
| Mots clés | |
| Service rendu | Un scénario assez élaboré faisant intervenir un utilisateur du futur système décrit en général un service que doit rendre le système à cet utilisateur. Cette rubrique permet de le décrire. |
| Agents intervenants | Enumération des agents intervenants dans le scénario ou l'épisode. |
| *Informations de création* | |
| Fournisseurs d'information | L'ensemble des personnes ayant apporté l'information nécessaire à la création du scénario ou de l'épisode. Pour chaque personne, on connaît son nom et un moyen de la contacter facilement. |
| Analyste | L'ensemble des personnes ayant formalisé et écrit les scénarios. Pour chaque personne on connaît son nom et un moyen de la contacter. |
| Date de dernière modification | La date de la dernière modification ayant été apportée au scénario ou à l'épisode. |
| Version | Une indication sur la version du scénario. Cette indication est nécessaire pour les traces et doit également comporter la mention 'en cours d'élaboration' ou 'terminé'. |
| Trace | Un ensemble de noms de scénarios ou d'épisodes ainsi que les versions de ceux-ci permet d'indiquer les étapes parcourues pour arriver à ce scénario ou à cet épisode et de retrouver facilement les produits intermédiaires. |
| *Documentation supplémentaire* | |
| Dessin de l'environnement | Le ou les dessins facilitant la compréhension du scénario, souvent en explicitant l'environnement. |
| Références | Toutes les références bibliographiques des ouvrages consultés pour comprendre et élaborer le scénario ou épisode. |
| Spécification ALBERT | Une référence vers la spécification ALBERT définissant le vocabulaire relatif à ce scénario. Cette référence doit aussi comprendre la version de la spécification. |
| Notes | Des notes supplémentaires intéressantes pour la compréhension du scénario ou de l'épisode. |

ii) Exemple

Dans le cadre du problème du carrefour, nous pouvons donner les informations relatives au scénario d'un passage 'normal' dans le carrefour. Nous représentons ces informations sous forme tabulaire.

| **Rubrique** | **Contenu** |
|---|---|
| *Spécification du scénario* | |
| Nom | Passage normal dans le carrefour. |
| Brève description | Une voiture rentre dans le carrefour et en sort en ayant tourné vers sa gauche. Tout se passe normalement comme dans la plupart des cas. |
| But | Ce scénario a pour but de montrer le comportement normal (90% des cas environ) d'un conducteur entrant dans le carrefour et sortant vers la gauche. Il donne également certains temps à titre indicatif pour donner une idée des délais rencontrés en temps normal. |
| Mots clés | passage normal, temps moyens. |
| Service rendu | Le service rendu au conducteur par le système dans le cadre de ce scénario est uniquement le service de base : assurer la sécurité du conducteur en empêchant les voitures venant de gauche ou de droite de croiser son chemin. |
| Agents intervenants | Voiture A, Feu F1. |
| *Informations de création* | |
| Fournisseurs d'information | Christophe Bongartz. |
| Analyste | Idem. |
| Date de dernière modification | 25/8/97. |
| Version | 1 |
| Trace | / |
| *Documentation supplémentaire* | |
| Dessin de l'environnement |  |

| Références | / |
|---|---|
| Spécification ALBERT | Carrefour - Annexe 1 |
| Notes | / |

## C. Le contexte du contenu du scénario

### i) Introduction

Le but de cette section est de définir un langage qui permet d'exprimer des aspects contextuels de l'exécution du scénario. On pourra exprimer des 'mises en situation' par rapport au scénario ou à l'épisode, des descriptions de 'situations en sortie' du scénario et des descriptions du comportement du reste du système pendant l'exécution du scénario ou d'un épisode.

Dans la littérature, nous n'avons pas trouvé la distinction entre contexte de l'exécution du scénario et contexte de création du scénario. Les deux sont généralement mélangés. A part les préconditions et postconditions écrites de manière relativement informelle, on ne trouve pas, dans la littérature, de langage spécialisé pour l'expression de ce genre de contexte.

Le langage défini ici sera formel. La raison principale est que les scénarios ALBERT peuvent être utilisés pour des animations de spécifications ALBERT à l'aide d'un outil d'animation. Vu que les informations véhiculées dans cette partie du contexte ont un grand intérêt lors d'une animation, il est intéressant de pouvoir les traiter et exploiter facilement sur une machine.

Les informations véhiculées sont intéressantes pour une animation d'une spécification ALBERT parce qu'elles permettent de simplifier considérablement deux problèmes rencontrés lors des animation : comment arriver à une situation initiale intéressante sans devoir 'exécuter la spécification jusque là' et comment réduire le nombre de questions liées à l'indétermination posé à l'utilisateur de l'animateur ?

Dans une première section, nous expliquerons comment nous sommes arrivés au langage défini dans la section d'après et quels sont ses bases intuitives. Nous terminerons avec un exemple.

### ii) Les principes

Cette section explique les raisons qui nous conduisent dans la construction du langage et elle explique les principes de base de ce langage.

Pour commencer, nous nous sommes posés la question de savoir ce qu'est une précondition dans le cadre d'un scénario ALBERT. Habituellement, une précondition d'une fonction est une condition qui doit être satisfaite pour que la fonction soit définie. Elle délimite le domaine d'application de la fonction.

Dans le cadre des scénarios c'est un peu différent. Bien sûr, on peut vouloir réduire le nombre de situations dans lesquelles le scénario a un sens. Mais en plus, il est intéressant de savoir dans quelle situation le scénario est intéressant. Il existe des situations dans lesquelles un certain scénario peut être appliqué mais où il n'apporterait pas beaucoup d'informations

intéressantes. Ce même scénario, placé dans une situation différente, peut apporter énormément d'informations.

Dans le cadre de l'exemple du carrefour, le scénario d'une voiture qui entre dans le carrefour et sort par le tronçon de route se trouvant à sa gauche est banal en cas de trafic normal. Cependant, mis dans une situation de trafic intense dans une certaine direction, ce scénario peut révéler des propriétés importantes du système de gestion du carrefour, notamment l'adaptation du temps vert pour les voies chargées.

La précondition d'un scénario sert donc de mise en situation au scénario.

Suite à cette constatation, nous avons décidé de séparer les scénarios de leur contexte. En effet, nous pouvons avoir plusieurs mises en situation différentes pour un même scénario et une même mise en situation peut être appliquée à différents scénarios.

La question suivante est : comment décrire une situation dans le cadre d'une spécification ALBERT ?

A priori, on pense à la description d'un certain état du système. Le problème est qu'il faudra décrire beaucoup d'éléments inutiles. En plus, il est intéressant de ne pas se limiter à un seul état mais plutôt tout un ensemble d'états partageant certaines propriétés intéressantes. Il est inintéressant de dire que dans une certaine voie on trouve d'abord la voiture A, puis la voiture B, puis la voiture C, etc. Ce qui est intéressant, c'est de pouvoir dire qu'on en trouve plus que 15, par exemple.

En plus, décrire uniquement l'état actuel élimine tout information sur ce qui s'est déjà passé. Certaines contraintes en ALBERT sont évaluées en tenant compte du passé. Il est donc intéressant de pouvoir exprimer des propriétés sur des choses qui se ont eu lieu dans le passé. Dans le cadre du carrefour, nous voulons pouvoir dire qu'avant de commencer un certain scénario, il n'y a pas eu beaucoup de voitures sur une certaine voie et que ce n'est que depuis 2 minutes que cette voie est surchargée.

Finalement, il nous restait à remarquer que, quand on ne dit rien, toutes les situations possibles sont acceptables et intéressantes pour le scénario. Chaque fois que nous décrivons une partie des situations intéressantes, nous restreignons les possibilités. Nous écrivons des contraintes sur les situations.

Nous avons ensuite remarqué que les contraintes de type 'State behavior' définies dans le langage ALBERT comportent plusieurs des ces propriétés recherchées. Elles permettent d'exprimer des propriétés historiques, se rapportent à l'état d'un agent, sont formelles et ont une capacité d'expression assez impressionnante.

Nous avons alors repris la syntaxe et la sémantique de base de ces contraintes à ceci près : elles ne se rapportent plus uniquement aux composants d'états d'un agent. En effet, il faut pouvoir exprimer des contraintes sur les composants d'état de tous les agents intervenant dans la spécification. Cependant, cela n'est pas encore assez. Il faut aussi pouvoir exprimer ce genre de contrainte sur les agents eux-mêmes, sur les occurrences de leurs actions et sur la communication entre agents.

Nous avons alors défini 4 tables contenant les informations pouvant être utilisées dans ces contraintes : la table des instances des agents, la table des composants d'état, la table des occurrences d'actions et la table des perceptions. La section suivante détaillera leur contenu exacte.

Vu que la précondition est une mise en situation par rapport au passé, la postcondition est la mise en situation par rapport au futur. Il est important de pouvoir décrire un ensemble de situations qu'on peut obtenir grâce à l'application d'un scénario à une situation initiale. Mais il est encore plus intéressant de se limiter aux propriétés vraiment intéressantes pour le futur.

Nous nous sommes rendus compte que ces propriétés peuvent s'exprimer avec le même langage. Les constructions nécessaires sont essentiellement les mêmes. La seule différence est que les contraintes dans la précondition se rapportent au passé alors que les contraintes de la postcondition se rapportent au futur.

Il nous restait donc uniquement la description du comportement du système à considérer dans le scénario. Cette description est d'un autre type puisqu'elle est d'une utilité un peu différente. Le type est différent puisqu'il ne faut plus résumer ce qui a conduit à une certaine situation (comme dans la précondition) mais plutôt résumer comment se comporte le système pendant un certain temps. Elle sert surtout dans le cadre d'une animation pour diminuer considérablement le nombre de questions que l'animateur doit poser à l'utilisateur pour éliminer les indéterminations.

Pour cela, nous avons besoin d'exprimer des probabilités associées à des occurrences d'actions ou à des perceptions, etc. Nous voulons par exemple pouvoir dire que durant le passage de la voiture dans le carrefour, des autres voitures arrivent en moyenne toutes les 15 secondes sur chaque autre voie. Les probabilités peuvent se rapporter à la communication entre deux agents. Cette communication est souvent non-déterministe. Un agent peut ou ne peut pas percevoir l'information. Nous pouvons vouloir dire qu'elle est perçue dans 95% des cas.

Afin de voir plus clair, nous avons identifié les endroits de non-déterminisme dans ALBERT :

1. le nombre d'instances de chaque acteur,
2. pas de valeur initiale pour un composant d'état,
3. le résultat d'une transmission d'informations entre acteurs,
4. les durées des actions,
5. le moment de début des actions, et
6. les expressions de la logique temporelle du premier ordre contenant un $\exists$ et/ou des connectives temporelles.

Le but étant de réduire l'indétermination, nous avons identifié les constructions prédicatives permettant de le faire. Pour les indéterminations de type 1, 2 et 5, il suffit d'écrire une contrainte en une ligne sur une des 4 tables. Pour les indéterminations 3 et 4, nous avons

fait des constructions syntaxiques permettant d'écrire les contraintes en une ligne. La construction syntaxique est équivalente à une expression du premier ordre se basant sur le 4 tables. L'indétermination 6 ne peut pas être enlevée en ajoutant des contraintes.

Nous pouvons donc uniquement grâce à des constructions syntaxiques exprimer les propriétés essentielles du comportement du système n'intervenant pas directement dans le scénario.

Le principe des constructions syntaxiques peut être utilisé pour apporter des améliorations futurs à ce langage. En effet, suite à des utilisations de ce langage sur des exemples concrets, nous pouvons trouver des ensembles de formules écrites régulièrement dans la spécification du contexte des scénarios. Ces formules peuvent alors être redéfinies syntaxiquement dans un but de simplification d'expression des contraintes. Nous pourrions ainsi obtenir un langage encore plus adapté à l'expression de situations en ALBERT.

Une autre amélioration peut être la classification de certaines constructions. On peut par exemple regrouper toutes les constructions servant à définir les instances des agents ou toutes les constructions servant à donner des valeurs aux composants d'états. Faute de temps et d'utilisation du langage, nous n'avons pas défini de telles rubriques.

Il nous reste encore un dernier point à fixer. Les contraintes de type 'State Behavior' sont évaluées de manière historique. Cela veut dire que l'évaluation de l'expression se rapporte uniquement aux faits passés par rapport au moment d'évaluation.

Dans notre cas, les contraintes exprimées dans le cadre de la précondition seront évaluées de manière historique à partir du début du scénario. Les contraintes dans le cadre de la postcondition sont évaluées vers le futur à partir de la fin du scénario et les contraintes décrivant le fonctionnement du reste du système sont évaluées de manière historique à partir de la fin du scénario.

Il y a donc deux moments importants à considérer : le début et la fin du scénario. Afin de pouvoir introduire ces moments dans les contraintes, introduisons deux variables représentant ces deux moments : `debut_scenario` et `fin_scenario`.

Nous avons ainsi obtenu un même langage adapté à la description des préconditions, postconditions et exécutions simultanées d'un scénario ALBERT.

### iii) Le langage

Comme expliqué dans la section précédente, nous utilisons la même syntaxe pour exprimer des contraintes de type 'state behavior', des préconditions et des postconditions. La différence se situe au niveau des éléments pouvant être référencés dans ces formules. Nous allons donc commencer à spécifier ces éléments avant de faire un rappel sur la syntaxe du langage.

Nous définissons les 4 variables suivantes : inst_agent, state_comp, act_occ et perceptions. Le type et la signification des éléments de la structure de chacune de ces variables est défini comme suit :

```
Inst_agent : table[STRING -> set{STRING}]
```

`Inst_agent` représente l'ensemble des instances d'agents ou de sociétés présentes lors d'une instanciation d'une spécification ALBERT. Il est indexé par le nom du type d'agent ou de société. Dans le cadre de l'exemple du carrefour, `Inst_agent['voiture']` fait référence à l'ensemble des instances de l'agent voiture. Dans le cas d'une déclaration d'un agent unique, il est clair que cet ensemble ne contient qu'un seul élément qui est le nom de l'agent.

```
State_comp : table[(STRING, STRING, STRING) -> DATATYPE]
```

`State_comp` représente l'ensemble de valeurs des composants d'état existant lors d'une instanciation d'une spécification ALBERT. Il est indexé par le triplet constitué du nom de l'agent, du nom de l'instance et du nom du composant d'état. Dans le cadre de l'exemple du carrefour, `State_comp[('voiture', 'Plaque 1Z483', 'position')]` fait référence à la valeur du composant d'état position de instance de l'agent voiture identifiée par 'Plaque 1Z483'.

`DATATYPE` peut être défini `par UNION[INTEGER, STRING, CHAR, REAL, X, Y, Z, ...]` où `X, Y et Z` sont des types déclarés dans la section Data Types de la spécification ALBERT concernée.

```
Act_occ : table[(STRING, STRING, STRING) -> seq{(INTEGER, REAL, REAL)}]
```

`Act_occ` représente l'ensemble des occurrences d'actions possibles lors d'une instanciation d'une spécification ALBERT. Il est indexé tout comme par le triplet nom de l'agent, nom de l'instance et nom de l'action. Les occurrences sont identifiées par un couple indiquant leur temps de début et leur temps de fin ainsi que par un entier pouvant être utilisé dans les perceptions. Ainsi, dans le cadre de l'exemple du carrefour, `Act_occ['voiture', 'Plaque 1Z483', 'entre_carrefour']` fait référence à la séquence des couples déterminant le début et la fin des instances de l'action donnée.

```
Perceptions : table [(STRING, STRING, STRING, INTEGER) -> set{STRING, STRING)]
```

`Perceptions` représente l'ensemble des agents percevant une certaine occurrence d'une action ou d'un composant d'état dans l'instanciation d'une spécification ALBERT. Dans le cas de perceptions d'actions, le quadruple nom de l'agent, nom de l'instance, nom de l'action et numéro d'instance (spécifié dans `Act_occ`) est identifiant de l'occurrence d'action. Dans le cas de la perception d'un composant d'état, le numéro d'instance vaut 0 et n'a aucune signification. Dans le cadre de l'exemple du carrefour, `Perceptions[('voiture', 'Plaque 1Z483', 'entre_carrefour', 1)]` dénote l'ensemble des noms d'agents et instances percevant l'action.

Rappelons maintenant la syntaxe des contraintes du style 'state behavior' :

```
<STATE-CONS> ::= [ [<ACTION-OCC-LIST>] ] <TLOGIC-EXPR>
```

<TLOGIC-EXPR> dénote une expression logique temporelle. La syntaxe exacte de <TLOGIC-EXPR> se trouve dans [MDLMA97] ou à l'adresse internet www.fun.cediti.be/~bju/labert_syntax.html.

Ces expressions sont des phrases écrites dans la logique des prédicats du premier ordre munie d'extensions temporelles. Ces extensions sont des connectives que l'on peut placer à la place d'un `pourtout` ou d'un `existe`. Voici quelques exemples de phrases écrites avec cette syntaxe :

```
[Pump(a, l)] Waterlever(a) ≥ 0 ∧ Gate = Open

[] InStock(Stock, a) ≥ SomF ¬InStock(Stock, a)

[Machine(_, _)] (Status = Down) ⇨ (Status = Down Until Stock = {})
```

Dans le cadre des expressions des pré- et postconditions, nous n'avons pas besoin de la partie `<ACTION-OCC_LIST>`. Une précondition et une postcondition sont donc des ensembles de phrases écrites dans la logique temporelle du premier ordre faisant référence aux éléments détaillés ci-dessus.

Les descriptions du comportement de la partie du système qui n'est pas directement concerné par le scénario utilisent le même formalisme. Elles disposent cependant de quelques constructions supplémentaires et de constructions types en fonction de l'indétermination qu'ils essayent d'enlever.

Nous allons rappeler les indéterminations pouvant être réduits et les constructions permettant de le faire :

1. le nombre d'instances de chaque acteur,
2. pas de valeur initiale pour un composant d'état,
3. le résultat d'une transmission d'informations entre acteurs,
4. les durées des actions,
5. le moment du début des actions.

Le nombre d'instances de chaque acteur peut être fixé par une contrainte du type `count['nom acteur'] = nombre d'acteurs.`

La valeur initiale d'un composant d'état peut être fournie par la contrainte `State_comp['nom agent', 'nom instance' ou variable, 'nom comp. d'état'] = valeur.`

L'indétermination pour les perceptions des actions peut être améliorée en combinant deux contraintes : une contrainte qui nous donne le nombre d'occurrences d'une action grâce à la variable act_occ et une fixant le nombre d'agents percevant cette action grâce à la variable perceptions. Pour faciliter ces contraintes, nous avons défini une nouvelle variable pouvant être utilisée et qui est dérivée des 4 variables existantes.

```
Proba_perc : table[(STRING, STRING, STRING) -> set{(STRING, STRING, INTEGER)}]
```

`Proba_perc` représente les probabilités de perception d'une action par les autres agents. Elle est indexée par le triplet nom de l'agent, nom de l'instance et nom de l'action. Le résultat

est un ensemble de triplets dont un élément représente le nom d'un agent, le nom de l'instance et la probabilité en pour-cent de la perception. `Proba_perc` est une construction syntaxique pouvant être traduite en contraintes sur les deux autres variables `act_occ` et `perceptions`.

`Proba_perc[x, y, z]` est défini comme suit :

```
(a, b, c) ∈ Proba_prec[x, y, z] ⇔

n = Count(act_occ[x, y ,z])

∧ m = Count(Y)

∧ (a, b, t) ∈ Y

∧ (a, b) ∈ perceptions[x, y, z, t]

∧ c= n/m
```

La durée d'une action peut être définie par une contrainte sur la variable `act_occ`. A nouveau, pour faciliter la construction de ces contraintes, nous proposons une construction syntaxique.

```
Act_dur : table[(STRING, STRING, STRING, INTEGRE) -> REAL]
```

`Act_dur` indique la durée d'une occurrence d'une action. Elle est indexée par le quadruple nom de l'agent, nom de l'instance, nom de l'action et numéro de l'instance. `Act_dur[a, b, c, d]` est défini comme suit :

```
Act_dur[a, b, c, d] = c ⇔

c = x - y ∧ (d, x, y) ∈ act_occ[a, b, x]
```

Les moments précis de début d'une action peuvent être définis par une contrainte du style :

```
Act_occ['nom agent', 'nom instance', 'nom_action'] = (numéro instance,
début action, fin action)
```

La sémantique de ce langage est bien sûr donnée par la sémantique de la logique temporelle des prédicats du premier ordre. Elle est aussi définie par les correspondances entre les 4 variables et les concepts ALBERT.

### iv) Exemple

Voici une mise en situation pour un certain scénario.

Le carrefour est constitué de 4 bras avec 1 seule bande par bras. Toutes les trajectoires sont possibles. Lorsqu'une voiture veut tourner à gauche, elle peut être bloquée par des voitures arrivant d'en face. Lorsqu'elle est a l'arrêt, les voitures venant de la même bande et voulant aller tout droit sont ralenties. Les feux sont couplés 2 à 2 en face. Cela veut dire que les feux se trouvant à l'opposé par rapport au carrefour ont toujours la même couleur. Les autres feux ont la couleur 'inverse' (i.e. rouge – vert).

Nous supposons une modélisation en ALBERT de ce problème comme elle est détaillée en annexe. Cet énoncé de mise en situation peut être traduit dans le langage défini ci-dessus. Des extraits du résultat sont donnés par le texte suivant. Le texte est assez long vu la situation décrite de manière très détaillée. En effet, plus la situation est particulière, plus il y a des contraintes à écrire.

```
'Bande A' ∈ Inst_agent['Bande']

'Bande B' ∈ Inst_agent['Bande']

'Bande C' ∈ Inst_agent['Bande']

'Bande D' ∈ Inst_agent['Bande']

Count(Inst_agent['Bande']) = 4

'Traj AB' ∈ Inst_agent['Trajectoire']

'Traj AC' ∈ Inst_agent['Trajectoire']
[...]

'Traj DC' ∈ Inst_agent['Trajectoire']

Count(Inst_agent['Trajectoire']) = 12

'Traj CA' ∈ State_comp[('Trajectoire', 'Traj AD',
    'traj_ralentissantes')]
[...]

'Traj AC' ∈ State_comp[('Trajectoire', 'Traj DC',
    'traj_ralentissantes')]

[...]
```

Voici quelques exemples de contraintes pouvant être exprimées par ce langage.

- Le nombre de voitures dans le carrefour est plus petit que 6 et les voitures A et B y sont :

```
Count(Inst_agent['Voiture']) ≤ 6

A ∈ Inst_agent['Voiture'])

B ∈ Inst_agent['Voiture'])
```

- Tous les feux sont passés au moins deux fois au rouge :

```
∀ x : count(Act_occ['Feu', x, 'passage_rouge']) ≥ 2
```

- A un moment donné, pendant les 5 dernières minutes, toutes les bandes étaient vides :

```
WithinP5' : ∀ x : Count(State_comp[('Bande', x, 'voitures')]) = 0
```

### 4.3.5 Récapitulatif

Nous disposons donc maintenant d'un langage permettant d'exprimer le contenu d'un scénario. Ce langage est graphique et se base sur le formalisme des MSC muni de quelques extensions. Il dispose d'un mécanisme de structuration facilitant la maîtrise des scénarios complexes. Il est adapté aux spécifications écrites en ALBERT et utilise le même vocabulaire.

Nous disposons également d'un langage formel de description de situations initiales et finales liées à un scénario. Ce langage est basé sur la logique temporelle des prédicats du premier ordre et dispose d'un ensemble de références et d'interprétations définies en fonction des spécifications ALBERT.

Finalement, nous disposons d'un langage servant de guide méthodologique à la gestion des scénarios. Ce langage est informel et définit des types d'informations liées à la spécification, à la création et à la documentation des scénarios.

Nous avons donc défini 3 langages différents pour exprimer 3 parties importantes des scénarios. Un premier pour le contenu, un deuxième pour les informations de manipulation et de gestion des scénarios et un dernier pour l'expression des préconditions, postconditions et la définition des exécutions parallèles.

Nous avons également parlé de l'indépendance du contenu d'un scénario par rapport au contexte formel. La description d'un situation peut s'appliquer à plusieurs scénarios et un scénario peut être utilisé dans plusieurs situations.

C'est justement cette considération qui nous pousse à préciser un peu notre vocabulaire. Dorénavant, nous allons parler de type de scénario lorsqu'il s'agit uniquement de la suite des actions du scénario, c'est-à-dire du contenu du scénario. Après avoir ajouté la mise en situation, c'est-à-dire la précondition, la postcondition et la description des exécutions du reste du système, nous parlerons de scénarios instanciés.

Le contexte informel se rapporte alors à une instance de scénario, à un type de scénario ou même à un contexte formel. Les rubriques adéquates sont remplies en fonction de l'association effective. Par exemple, le contexte informel lié à un type de scénario ne précisera pas nécessairement les rubriques du but ou du dessin de l'environnement alors qu'elles sont assez intéressantes dans le cadre d'un contexte informel d'une description de situation.

Le contexte informel est le plus intéressant pour un scénario instancié. C'est pour un scénario instancié qu'on a assemblé un type de scénario et une situation bien précise dans un certain but.

Le schéma à la Figure 37 résume la relation qui existe entre les 3 langages fournis pour les 3 parties des scénarios.

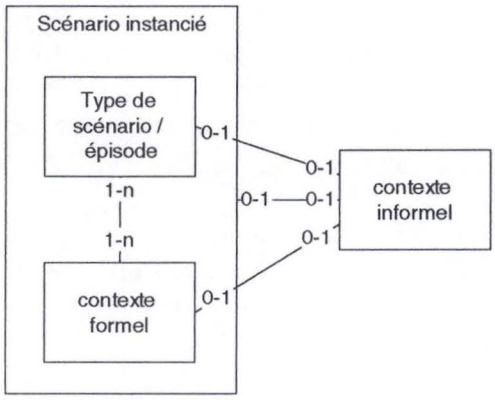



**Figure 37 - relations entre les trois parties de la description d'un scénario**

### 4.3.6 Un exemple complet

Le but de cette section est d'illustrer l'utilisation des 3 langages définis ci-dessus à travers un exemple concret choisi dans le cadre de l'étude de cas.

Rappelons que l'étude de cas spécifie un système devant permettre de commander n'importe quel type de carrefour réglé par des feux rouges. La spécification ALBERT liée à ce problème peut être consultée à l'annexe 1.

L'exemple décrit deux scénarios se rapportant à une même configuration de carrefour. Pour commencer, nous allons expliquer cette configuration plus en détail.

*Le carrefour considéré est une intersection de 3 tronçons de route. On parle aussi d'un carrefour à 3 bras. A leur point de rencontre, les 3 tronçons forment un 'T'. 2 des 3 tronçons disposent d'une bande d'approche et le troisième dispose de 2 bandes d'approche. Toutes les trajectoires imaginables sont possibles. Une bande dispose d'un système de détection de présence de voitures ayant une capacité de 3, c'est-à-dire il peut voir la présence de 3 voitures. Les feux F1 et F2 sont synchronisés entre eux. Ils ont chacun une contrainte d'exclusion avec le feu F3. La Figure 38 illustre la configuration du carrefour et donne des noms aux bandes, trajectoires et feux intéressants du carrefour.*

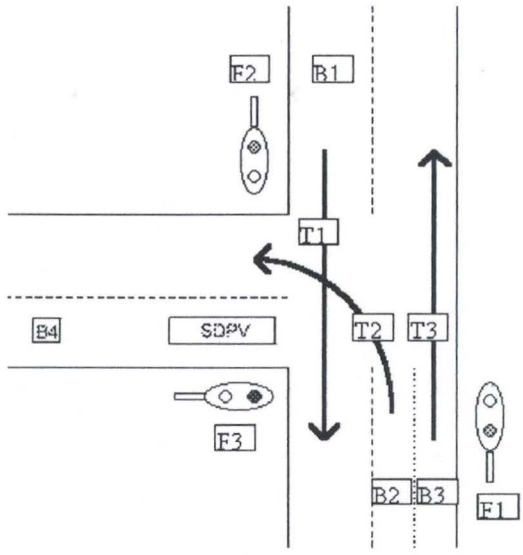



**Figure 38 - illustration de la configuration du carrefour**

Cette description fait partie de la mise en situation des 2 scénarios que nous allons décrire. Pour ne pas devoir répéter à chaque fois cette configuration, nous écrivons les contraintes associées à cette description de carrefour une seule fois et nous y ferons référence par après. Voici donc la précondition exprimant la configuration concrète du carrefour. Elle sera identifiée par le nom 'configuration carrefour'. Il s'agit bien sûr d'une précondition, d'une postcondition et d'une 'description du système lors de l'exécution du scénario' en même temps. Les remarques seront précédées d'une double barre oblique (//).

// Il y a 3 feux, F1, F2, F3

```
'F1' ∈ Inst_agent['Feu']
'F2' ∈ Inst_agent['Feu']
'F3' ∈ Inst_agent['Feu']
Count(Inst_agent['Feu']) = 3
```

// Il y a 4 bandes et 6 trajectoires. Dans le cadre de cet exemple, on n'a pas besoin de savoir lesquels.

```
Count(Inst_agent['Bande']) = 4
Count(Inst_agent['Trajectoire']) = 6
```

// La bande associée au feu F3 dispose d'un SDPV d'une capacité de 3. Il est intéressant de pouvoir reconnaître cette bande.

```
∃ t ∈ Inst_agent['Trajectoire'] ∧
  ∃ b ∈ Inst_agent['Bande'] ∧
  State_comp['Trajectoire', t, 'Bande_origine'] = b ∧
  State_comp['Trajectoire', t, 'Feu_regie'] = 'F3' ∧
  State_comp['Bande', b, 'Capacite_SDPV' = 3 ∧
  b = 'B4'
```

// Les feux F1 et F2 sont synchronisés entre eux

```
'F1' ∈ State_comp['Feu', 'F2', 'Synchronise']
```

```
'F2' ∈ State_comp['Feu', 'F1', 'Synchronise']
```

// Les feux F1 et F2 ont une exclusion avec F3 et vice-versa

```
'F1' ∈ State_comp['Feu', 'F3', 'Exclusion']
'F2' ∈ State_comp['Feu', 'F3', 'Exclusion']
'F3' ∈ State_comp['Feu', 'F1', 'Exclusion']
'F3' ∈ State_comp['Feu', 'F2', 'Exclusion']
```

Le premier scénario d'exemple a pour but d'expliquer la réaction du système suite à la détection de voitures en attentes devant le feu F3. Voici le texte informel décrivant ce scénario.

*Le feu F3 vient de passer au rouge. Aucune voiture n'attend devant ce feu. Les feux F1 et F2 sont verts et des voitures y passent à une dizaine par demi-minute. 30 secondes après que le SDPV ait détecté la 3ème voiture en attente devant le feu F3, les feux F1 et F2 passent à l'orange et puis au rouge. Quelques instants après, le feu F1 passe au vert. Après que le SDPV n'ait plus détecté de voitures depuis plus que 5 secondes, et au maximum après 2 minutes après le passage au vert, le feu F3 passe à nouveau à l'orange et puis au rouge.*

Nous rencontrons d'abord une description du contexte précis de ce scénario. Dans cette description, nous rencontrons aussi des description du fonctionnement de la partie du système qui ne nous intéresse pas. Ces éléments sont représentés grâce au langage formel de description du contexte du scénario ci-dessous.

Voici d'abord la précondition qui sera identifiée par le nom 'Bande B4 vide et feu F3 rouge'.

// Le feu F3 vient de passer au rouge

```
(1, -1 sec, -1 sec) ∈ Act_occ['Feu', 'F3', 'passage_rouge']
debut_scenario = 0
```

//Aucune voiture n'attend devant ce feu

```
{} = State_comp['Bande', 'B4', 'Voitures']
```

// Les feux F1 et F2 sont verts

```
State_comp['Feu', 'F1', 'Couleur'] = 'Vert'
State_comp['Feu', 'F2', 'Couleur'] = 'Vert'
```

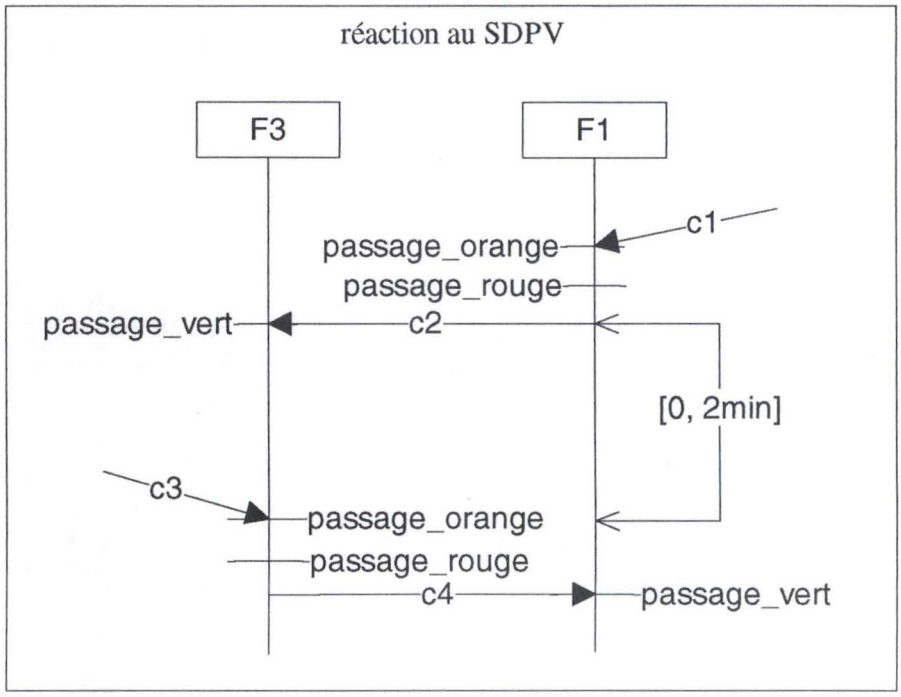
Voici la contrainte décrivant le fonctionnement du reste du système durant l'exécution du scénario. Elle sera identifiée par le nom 'Flux normal par F1 et F2'.

// Les voitures passent par les feux F1 et F2 à une raison de 10 par minute environ.

```
X ⊂ Dom(Act_occ) ∧
   ∀ (a, b, c) ∈ X : a = 'Voiture' ∧
   c = 'Entre' ∧
   State_comp['Voiture', b, 'Intention'] = t
   ∧ State_comp['Trajectoire', t, 'Feu_regie'] = 'F1' ∧
   Act_occ[a, b, c] = (d, e, f) ∧
   debut_scenario ≤ e ≤ f ≤ fin_scenario ∧
   #X = 10
```

```
X ⊂ Dom(Act_occ) ∧
   ∀ (a, b, c) ∈ X : a = 'Voiture' ∧
   c = 'Entre' ∧
   State_comp['Voiture', b, 'Intention'] = t
   ∧ State_comp['Trajectoire', t, 'Feu_regie'] = 'F2' ∧
   Act_occ[a, b, c] = (d, e, f) ∧
   debut_scenario ≤ e ≤ f ≤ fin_scenario ∧
   #X = 10
```

Le reste de la description concerne le contenu du scénario. Il est représenté ci-dessous grâce à un bMSC. Cet épisode (tout scénario est aussi un épisode) sera identifié par le nom 'réaction au SDPV'.



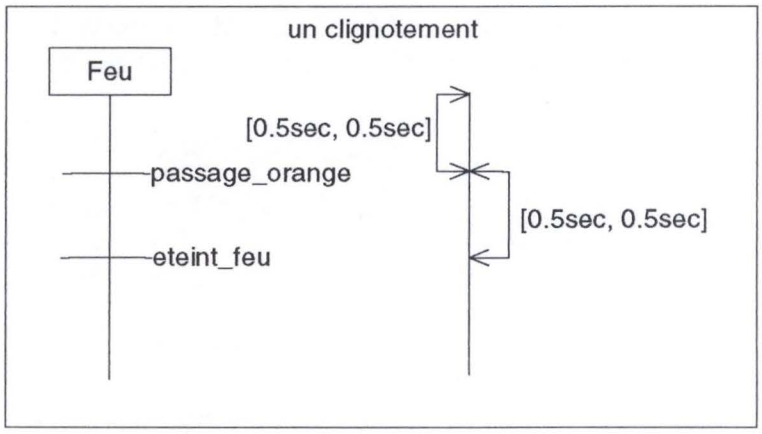

```
c1 : State_comp['Bande', 'B4', 'Nbr_voit_detectes'] ≥ 3 since 30 sec
c2 : Couleur = Rouge
c3 : State_comp['Bande', B4', Nbr_voit_detectes'] = 0 since 5 sec
c4 : Couleur = Rouge
```

Comme les flèches c1 et c2 viennent de l'extérieur du schéma, les libellés de ces flèches se rapportent à la partie du système qu'on n'observe pas dans ce scénario.

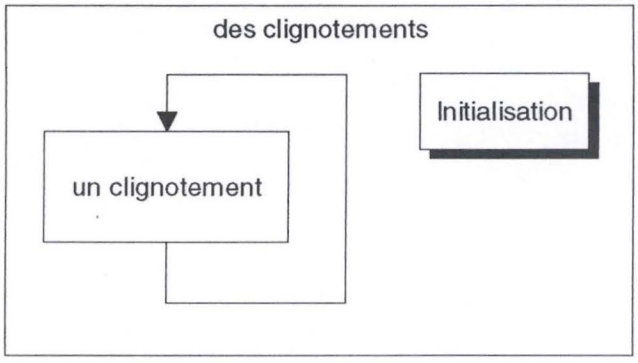
Le deuxième scénario a pour but de montrer d'une part l'utilité des épisodes d'interruption et d'autre part de découvrir un oubli dans la modélisation du carrefour : la panne du système. Voici l'énoncé informel du scénario.

*Pour n'importe quel carrefour et à n'importe quel moment, une panne peut être détectée par le système et les feux se mettent à clignoter. Lorsque les feux clignotent, la lampe orange s'allume et s'éteint à un rythme de un changement toutes les demi secondes. La fin de cette phase peut être obtenue par une initialisation du système.*

Nous supposons qu'il existe un scénario décrivant l'initialisation. Ce scénario est identifié par le nom 'initialisation'. Le scénario s'applique à n'importe quel carrefour et nous travaillerons donc avec le type 'Feu' au lieu de travailler avec des instance de ce type. Voici le schéma de type bMSC décrivant l'épisode d'allumage et d'éteinte du feu orange. Il est identifié par le nom 'un clignotement'.




Ce clignotement se répète infiniment jusqu'à l'interruption par l'épisode 'initialisation'. Le hMSC suivant représente cette partie du scénario identifiée par le nom 'des clignotements'.




Cet épisode 'des clignotements' doit alors figurer sur tous les scénarios relatifs aux carrefours qui peuvent être interrompus par une panne du système.

Il nous reste à décrire le contexte de création associée à ces différents scénarios et les mises en situation. Nous en écrirons 3 :

- un pour la description de la composition du carrefour,
- un pour la description du scénario 'réaction du SDPV' associé aux trois mises en situation 'configuration du carrefour', 'Bande B4 vide et feu F3 rouge' et 'Flux normal par F1 et F2',
- et un pour la description du scénario 'des clignotements'.

Nous représentons les différentes rubriques sous forme tabulaire.

| Nom rubrique | Description |
|---|---|
| *Spécification du scénario* | |
| Nom | Configuration carrefour. |
| Brève description | Contexte de scénario décrivant un carrefour concret à 3 bras. |
| But | Définir les éléments importants d'un certain carrefour à 3 bras. |
| Mots clés | |
| Service rendu | |
| Agents intervenants | N/A. |
| *Informations de création* | |
| Fournisseurs d'information | Christophe Bongartz. |
| Analyste | Christophe Bongartz. |
| Date de dernière modification | 29/8/97. |
| Version | 1 |
| Trace | |
| *Documentation supplémentaire* | |
| Dessin de l'environnement | Cfr. ci-dessus. |
| Références | |
| Spécification ALBERT | Spécification 'Carrefour' en annexe 1. |
| Notes | |

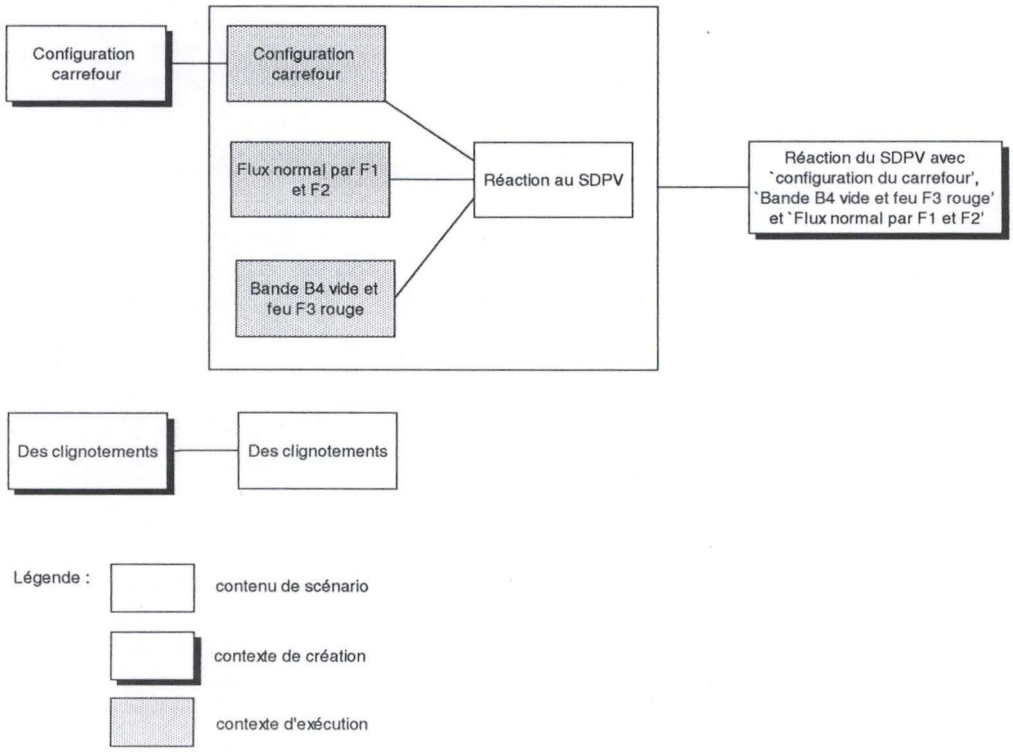
| Nom rubrique | Description |
|---|---|
| *Spécification du scénario* | |
| Nom | Réaction du SDPV avec 'configuration du carrefour', 'Bande B4 vide et feu F3 rouge' et 'Flux normal par F1 et F2' |
| Brève description | Lorsque le SDPV de la bande B4 détecte le maximum de voitures pendant 30 secondes, le feu F3 devient vert. Lorsqu'il ne détecte plus rien pendant 5 secondes, le feu redevient rouge. |
| But | Un premier but est de montrer la réaction du système face à la détection de voitures sur la bande B4. Un deuxième but est de fixer les temps intervenant dans ces réactions |
| Mots clés | |
| Service rendu | |
| Agents intervenants | N/A. |

| *Informations de création* | |
|---|---|
| Fournisseurs d'information | Christophe Bongartz. |
| Analyste | Christophe Bongartz. |
| Date de dernière modification | 29/8/97. |
| Version | 1 |
| Trace | |
| *Documentation supplémentaire* | |
| Dessin de l'environnement | Cfr. ci-dessus. |
| Références | |
| Spécification ALBERT | Spécification 'Carrefour' en annexe 1. |
| Notes | |

| **Nom rubrique** | **Description** |
|---|---|
| *Spécification du scénario* | |
| Nom | Des clignotements |
| Brève description | Tous les feux clignotent sur la couleur orange jusqu'à leur interruption par le scénario 'initialisation' |
| But | Le premier but est de montrer le fonctionnement des interruptions de scénarios et le deuxième est de montrer un oubli dans la spécification ALBERT. |
| Mots clés | |
| Service rendu | |
| Agents intervenants | N/A. |
| *Informations de création* | |
| Fournisseurs d'information | Christophe Bongartz. |
| Analyste | Christophe Bongartz. |
| Date de dernière modification | 29/8/97. |
| Version | 1 |
| Trace | |
| *Documentation supplémentaire* | |
| Dessin de l'environnement | Cfr. ci-dessus. |

| Références | |
|---|---|
| Spécification ALBERT | Spécification 'Carrefour' en annexe 1. |
| Notes | |

Pour résumer la situation, le schéma ci-dessous montre la relation existant actuellement entre les différentes descriptions de contenus et contextes des deux scénarios.




Ceci termine l'exemple.

# 5. Conclusion

En conclusion nous apportons une critique à ce travail en évaluant ses apports et ses limites et nous donnons des indications sur des améliorations des langages définis.

## *5.1 Les apports et limites du travail*

### 5.1.1 Les apports principaux

L'exploitation de scénarios est une technique de plus en plus utilisée dans le domaine de l'ingénierie des besoins. Elle aide à découvrir des propriétés intéressantes du domaine d'application et du système à spécifier. Elle permet aussi la validation des spécifications par rapport au fonctionnement de l'environnement du système. Dans quelques cas, un ensemble de scénarios bien choisis *est* la spécification du système.

L'augmentation de l'utilisation de scénarios provoque également une augmentation des langages permettant de les représenter. Chaque entreprise voulant travailler avec des scénarios essaye d'adapter différents formalismes de représentation à ses besoins. Malgré des efforts de standardisation de certaines de ces représentations (ex. : MSC par l'ITU-T), on ne trouve pas de consensus accepté par tout le monde.

L'intérêt de ce travail est d'essayer de clarifier un peu les points communs à toutes les représentations et d'y mettre un peu de structure. Par sa distinction entre la partie contenu, la partie contexte d'exécution et la partie contexte de création, et par son identification des éléments importants de chacune de ces parties, ce travail donne un guide conceptuel pour la représentation des scénarios.

Des travaux sont actuellement en cours pour développer un animateur de spécifications écrites en ALBERT. Les scénarios jouent un rôle intéressant aussi bien en amont qu'en aval d'une animation. Grâce à un contexte d'animation et à des scénarios prédéfinis, les interactions entre l'animateur et l'utilisateur peuvent être fortement réduits.

Les langages définis dans ce travail ont étés adaptés à une utilisation de scénarios dans le cadre de spécifications ALBERT. La partie du contexte d'exécution offre des écritures spécialement adaptées aux animations ALBERT.

Dans le cadre du langage exprimant le contenu des scénarios, la redéfinition de la sémantique des 'flèches' indiquant l'échange d'un message entre acteurs dans une représentation sous forme de MSC a été faite en vue d'expression de scénarios employant des concepts d'ALBERT.

### 5.1.2 Les apports secondaires

A part la définition des langages d'expression des scénario, ce travail fait un rappel sur l'ingénierie des besoins en identifiant ses résultats, les processus menant à ces résultats et les problèmes rencontrés.

De même, il définit des concepts proches des scénarios comme ceux des « use case », prototype, animation et simulation, et souligne l'utilité des scénarios ainsi que les problèmes liés à leur construction. Il est intéressant de voir les dimensions importantes de la représentation des scénarios.

Tout cela ainsi que les états de l'art en termes de moyens de représentation de scénarios forme une lecture intéressante pour toute personne s'initiant aux scénarios.

Dans le cadre de l'animateur de spécifications ALBERT, ainsi que dans des buts de validations du langage ALBERT, l'étude de cas apporte des renseignements intéressants. Elle montre entre autres la complexité de la modélisation de certaines parties en ALBERT. Ainsi, nous avons pu nous rendre compte que le langage ALBERT montre des faiblesses pour la modélisation de systèmes génériques. La spécification examinée pour l'étude de cas s'applique à n'importe quelle configuration de carrefours. ALBERT est mieux adapté à modéliser un carrefour concret.

### 5.1.3 Les limites

Définir complètement un langage d'expression de scénarios et lui associer une sémantique précise n'est sûrement pas le but d'un travail comme celui-ci. Rien qu'au niveau des contraintes temporelles, cet objectif ne peut être atteint. Notre objectif était cependant d'atteindre un minimum de rigueur dans la construction du langage.

Le résultat obtenu n'atteint peut être pas le niveau de rigueur initialement voulu. Beaucoup de questions liés à l'interprétation de certaines constructions syntaxiques restent ouvertes dues à une spécification insuffisante.

La sémantique liée au langage a été définie de manière informelle et assez intuitive. Nous pensons que la définition intuitive permettra d'expérimenter les langages et d'apporter facilement les changements nécessaires à une meilleure utilisation. C'est uniquement après avoir confirmé les interprétations intuitives par plusieurs applications concrètes qu'il est temps de définir une sémantique rigoureuse.

Les idées exposées dans ce travail se basent surtout sur la littérature et sur des réflexions théoriques cherchant à trouver des solutions adéquates. Nous n'avons que peu d'expériences concrètes de l'ingénierie des besoins et particulièrement de la technique des scénarios. Même si l'étude de cas apporte certaines indications intéressantes, elle reste un problème théorique sans contrepartie pratique réelle. Le système de gestion de carrefours ne sera jamais implémenté sur base des spécifications fournies dans ce travail. Les scénarios ne viennent pas d'un expert du domaine ou d'un utilisateur potentiel du système mais bien de ceux qui ont défini le langage.

Tout cela fait que l'intérêt du travail doit être plus vu au niveau théorique qu'au niveau pratique. La division de l'expression des scénarios en 3 parties, les décisions prises lors de

l'évaluation des possibilités et la manière d'aborder le problème sont les vrais atouts de ce travail, bien plus que le langage lui-même.

## 5.2 L'avenir

Comme on peut le remarquer dans le paragraphe précédent, ce travail n'est pas terminé. Il a couvert une étape importante vers la représentation structurée des scénarios ALBERT.

La suite du travail peut être dérivée des remarques sur ses limites. Pour vraiment voir l'intérêt du langage du contenu, du langage du contexte d'exécution et du langage du contexte de gestion des scénarios, il faut les expérimenter.

Une étude de cas réelle permettra de détecter les problèmes que nous n'avons pas remarqué lors de nos réflexions théoriques. Elle aidera aussi à mieux saisir la sémantique par rapport aux spécifications ALBERT. Une première suite serait donc d'expérimenter largement les 3 langages de description de scénarios.

Suite aux conclusions tirés à partir de ces expérimentations, on sera capable de définir les langages de manière plus formelle et plus rigoureuse. Cela s'applique bien sûr uniquement au langage du contenu et au langage du contexte d'exécution.

Suite à la formalisation du langage, des règles de vérification et de validation beaucoup plus poussées peuvent être définies. Finalement, un outil peut soutenir l'analyste dans l'exploitation du langage.

# 6. Bibliographie

## 6.1 L'ingénierie des besoins


**[Hofmann 93]**

Hubert F. Hofmann, *Requirements Engineering – A Survey of Methods and Tools*, Institut für Informatik der Iniversität Zürich, Nr. 93.05, Mars 1993.

**[Hayes 90]**

I.J. Hayes, C.B. Jones, "Specifications are not (necessarily) executable", *Software Engineering Journal*, Vol 4 No 6, pp330-338, Janvier 1990.

**[Jackson 96]**

Michael Jackson, Pamela Zave, *Four dark corners of requirements engineering*, Février 1996, version distribuée pour demandes de corrections.

**[Jacobson]**

Ivar Jacobson, *Use Cases in Large-Scale Systems*, Objectory Corp.

**[Loucopoulos 95]**

Pericles Loucopoulos, Vassilios Karakostas, *Sytem requirements engineering*, McGraw-Hill, London, 1995

**[Maiden 93]**

N.A.M. Maiden, A.G. Sutcliffe, "Requirements Engineering by Example : an Empirical Study", *Proceedings of First International Symposium on Requirements Engineering*, IEEE Computer Society Press, Janvier 1993, p104-112.

**[Morris 93]**

Derrick Morris, Boris Tamm, *Concise Encyclopedia of Software Engineering*, Pergamon Press, 1993, p286-293

**[Wohlin 94]**

Claes Wohlin, Björn Regnell, Anders Wesslén and Henrik Cosmo, "User-Centred Software Engineering – A Comprehensive View of Software Development", *Proceedings of Nordic Seminar on Dependable Computing Systems*, Lyngby, Denmark, Août 1994.

**[Wieringa 97]**

Roel Wieringa, "Advanced Object-Oriented Requirements Specification Methods – Tutorial", *International Symposium of Requirements Engineering*, 6-10 janvier 1997, Annapolis (USA)

## 6.2 ALBERT

**[DDZ 96]**

Philippe Du Bois, Eric Dubois, Jean-Marc Zeippen, "On the Use of Formal R.E. Language - The Generalized Railroad Crossing Problem", *Proceedings of the Third International Symposium on Requirements Engineering (RE'97)*, Janvier 5-8 1997, Annapolis, Maryland (USA)

**[Du Bois 95]**

Philippe Du Bois, *The ALBERT Language – On the Design ans Use of a Formal Specification Language for Requirements Analyses*, Institut d'Informatique – FUNDP, Namur, 1995

**[MDLMA97]**

E. Dubois, *Méthodologie de développement de logiciel: matière approfondie*, Notes de cours, Institut d'Informatique, 1997

## 6.3 Les « use case » et scénarios

**[Abdallah 97]**

Hanêne Ben-Abdallah, Stefan Leue, *Expressing and Analysing Timing Constraints in Message Sequence Charts*, Technical Report 97-04, Electrical and Computer Engineering University of Waterloo, Canada, Avril 1997

**[Andersson 95]**

Michael Andersson, Johan Bergstrand, *Formalizing Use Cases with Message Sequence Charts*, Master thesis, Departement of Communication Systems at Lund Institute of Technology, May 1995

**[Benner et al 93]**

Kevin M. Benner, Martin S. Feather, W. Lewis Johnson adn Lorna A. Zorman, "Utilizing Scenarios in the Software Development Process", *Information System Development Process, IFIP Transactions A-30*, Elsevierr Science Publishers, September 1993.

**[Berard]**

Edward V. Berard, *Be Careful with Use Cases*, The Object Agency, http://www.toa.com/online/pdf/pdf.html

**[CREWSD1-I1.1]**

C. Rolland, C. Ben Achour, C. Cauvet, J. Ralyté, A. Sutcliffe, N.A.M. Maiden, M.Jarke, P. Haumer, K. Pohl, E. Dubois, P. Heymans, "A Proposal For A Scenario Classification Framework", *Esprit 4th Framework Programme CREWS 21.903, Cooperative Requirements Engineering With Scenarios*, Deliverable D1-I1.1.

**[CREWSD1-I1.2]**

E. Dubois, P. Heymans, M. Jarke, K. Pohl, K. Weidenhaupt, A. Sutcliffe, N.A.M. Maiden, "Formal Scenario Representation : Initial Version and Examples", *Esprit 4th Framework Programme CREWS 21.903, Cooperative Requirements Engineering With Scenarios*, deliverable D1-I1.2

**[DDD 94]**

Eric Dubois, Philippe Du Bois, Frédéric Dubru, "Animating Formal Requirements Specifications of Cooperative Information Systems", *Proceedings of the Second International Conference on Cooperative Information Systems*, CoopIS-94, May 17-20 1994.

**[Heymans]**

Patrick Heymans, *Some thoughts about the animation of formal specifications written in the ALBERT language*, Institut d'Informatique – FUNDP, Namur.

**[Jacobson 94a]**

Ivar Jacobson, "Use Cases and Objects", *ROAD*, Volume 1 Number 4, Novembre – Décembre 1994, p8-10.

**[Jacobson 94b]**

Ivar Jacobson, "Basic Use-Case Modeling", *ROAD*, Volume 1 Number 2, juillet – août 1994, p15-19

**[Jacobson 95a]**

Ivar Jacobson, Dave Thomas, "Extensions", *ROAD*, Volume 1 Number 5, Janvier – Février 1995, p7-9.

**[Jacobson 95b]**

Ivar Jacobson, Stefan Byland, Patrik Jonsson, Staffan Ehneboom, "Using contracts and use cases to build plugable architectures", *JOOP*, Mai 1995, p18-24;76.

**[Jacobson 95c]**

Ivar Jacobson, "Formalizing use-case modeling", *JOOP*, juin 1995, p10-14.

**[Kimbler]**

Kristofer Kimbler, Daniel Sobirk, *Use Case Driven Analyses of Feature Interaction*, Departement of Communication Systems, Lund Institute of Technology, Sweden.

**[Mendoza 95]**

Victor M. Mendoza-Grado, *Formal Verification of Use Cases*, AT&T Bell Labs, Août 1995

**[Potts 94]**

Colin Potts, Kenji Takahashi, Annie Anton, "Inquiry-Based Scenario Analyses of System Requirements", *GIT-CC-94/14*, Janvier 1994

**[Regnell 95]**

Björn Regnell, Kristofer Kimbler, Anders Wesslén, "Improving the Use Case Approach to Requirements Engineering", *IEEE 0-8186-7017-7/95*, p40-47.

**[Regnell 96]**

Björn Regnell, Michael Andersson, Johan Bergstrand, "A Hierarchical Use Case Model with Graphical Representation", *Proceedings of ECBS'96, IEEE International Symposium and Workshop on Engineering of Computer Based Systems*, Mars 1996.

**[Rumbaugh 94]**

James Rumbaugh, "Getting started – Using use cases to capture requirements", *JOOP*, septembre 1994, p88-12;23.

**[SEED 95]**

SEED-Pro Team, *SEED-Pro – Use Case Model & Requirements Analyses*, Carnegie Mellon University, Pittsburgh PA, juin 1995.

**[Some et al]**

Stéphane Some, Rachida Dssouli, Jean Vaucher; *Un cadre pour l'ingénierie des exigences avec des scénarios*, Département d'Informatique et de Recherche Opérationnelle – Université de Montréal.

**[Vemulapalli]**

Chandra Vemulapalli, *A Use Case FAQ*, Advanced Software Technologies Group, http://www.rational.com/ot/uml.html

## 6.4 Autres ouvrages

**[Bodart 93]**

F. Bodart, Y. Pigneur, *Conception assistée des systèmes d'information – méthode modèles outils*, Masson, Paris, 1993

**[Comm97]**

N. Habra, *Pratiques de communication*, Notes de cours, Institut d'Informatique, 1997

**[Dubru 93]**

Fédéric Dubru, *Prototypage de spécifications formelles des besoins*, mémoire, Institut d'Informatique - FUNDP, Namur, juin 1993

**[ESEC 91]**

A. van Lamsweerde, A. Fugetta, "Lecture Notes in Computer Science", *ESEC '91*, Springer Verlag, 1991, p485-488

**[GTI97]**

Yves Pigneur, *Gestion des technologies de l'information*, Notes de cours, Institut d'Informatique, 1997

**[IHM96]**

François Bodard, Jean Vanderdonck, *Conception d'interfaces homme/machine*, Notes de cours, Institut d'Informatique, 1996

**[Jacobson 92]**

Ivar Jacobson, Magnus Christerson, Patrik Jonsson, Gunnar Övergaard, *Le génie logiciel orienté objet*, Addison-Wesley, Paris, 1993.

**[Jacobson 95]**

Ivar Jacobson, Maria Ericsson, Agneta Jacobson, *The Object Advantage – Business Process Reengineering with Object Technology*, ACM Press - Addison-Wesley, Wokingham, 1995.

**[MDL96]**

E. Dubois, *Méthodologie de développement de logiciel*, Notes de cours, Institut d'Informatique, 1996

**[Micro Robert 78]**

*Micro Robert – Dictionnaire du français primordial*, Le Robert S.N.L., Paris, 1978

**[Selic 94]**

Bran Selic, Garth Gullekson, Paul T. Ward, *Real-Time Object-Oriented Modeling*, Wiley, 1994

**[UML]**

Rational Software Corporation, *UML Notation Guide, UML Semantics*, dernière version disponible à l'adresse www.rational.com

# 7. Annexes

## 7.1 Annexe 1 - Spécification ALBERT du carrefour

L'étude de cas énoncé à la section 2.7 et modélisé à la 2.8.5 a donné lieu à une spécification en ALBERT du problème du carrefour. Cet annexe donne le texte complet de cette spécification.

Dû à des problèmes techniques, tous les commentaires sont dépourvus d'accents. Nous vous prions de nous en excuser.

**SPEC CARREFOUR**

CONSTRUCTED TYPES

```
| Une des 3 couleurs d'un feu
```

*COULEUR=ENUM[Vert, Orange, Rouge]*

```
| L'enumeration des differentes positions d'une voiture par rapport au carrefour.
```

*POSITION_VOIT=ENUM[Loin_avant, Devant_feu, Dans_carref, Loin_apres]*

```
| Utilise dans le state component Feu.synchronise
```

*SYNCHRO_FEU=CP[Feu: Feu, Coul: COULEUR]*

**SOCIETY CARREFOUR**

*(Voiture)))*

*(Feu)))*

*(Bande)))*

*(Trajectoire)))*

*(Autre_carrefours)))*

**AGENT CARREFOUR.VOITURE**

DECLARATION

```
| Represente la classe des voitures
```

STATE COMPONENTS

Retient la position d'une voiture par rapport au carrefour

*Position* instance-of *POSITION_VOIT*

Decrit l'intention qu'a le conducteur relatif a sa destination.

Elle est exprimee sous forme d'une reference a une trajectoire que va emprunter le conducteur.

**Intention* instance-of instance-of*Carrefour.Trajectoire* → *Carrefour.Trajectoire Carrefour.Bande*

Indique si la voiture peut sortir à une vitesse normale du carrefour ou si elle est genee par d'autres voitures.

Lorsqu'elle est genee, l'action de sortie du carrefour prend plus de temps.

*Ralentie* instance-of *BOOLEAN* derived-from *Intention Position*

### ACTIONS

Le moment auquel la voiture devient interressante pour le fonctionnement du carrefour.

**Approche* → *Carrefour.Bande*

Le franchissement de la ligne delimitant le carrefour.

*Entre* → *Carrefour.Bande Carrefour.Trajectoire*

Le moment de sortie du carrefour (passage de la ligne delimitant le carrefour)

*Sort* → *Carrefour.Trajectoire*

## BASIC CONSTRAINTS

### DERIVED COMPONENTS

La voiture est ralentie si elle est dans le carrefour et si il y a au moins une voiture dans une des trajectoires ralentissantes par rapport à sa trajectoire.

$Ralentie \triangleq Position = Dans\_carref$
$\Lambda\ (\exists\ t : t \in Intention.Traj\_ralentissantes\ \Lambda\ Count(t.Voitures) > 0)$

### INITIAL VALUATION

Au depart, toute voiture se trouve loin du carrefour

*Position* = *Loin_avant*

## DECLARATIVE CONSTRAINTS

### ACTION DURATION

Le temp que met la voiture pour entrer dans le carrefour est au minimum 2 secondes

*Entre > 2 sec*

> Le temps qu'il faut a la voiture pour entrer dans le carrefour est maximum 4 secondes

*Entre < 4 sec*

> Lorsque la voiture n'est pas ralentie par d'autres voitures, il lui faut minimum 2 secondes pour sortir

*Sort > 2 sec* with *¬(Ralentie)*

> Lorsque la voiture n'est pas ralentie par d'autres voitures, il lui faut au maximum 4 secondes pour sortir

*Sort < 4 sec* with *¬(Ralentie)*

> Lorsque la voiture est ralentie, il lui faut au minimum 5 secondes pour sortir

*Sort > 5 sec* with *Ralentie*

> Lorsque la voiture est ralentie il lui faut au maximum 8 secondes pour sortir du carrefour

*Sort < 8 sec* with *Ralentie*

## OPERATIONAL CONSTRAINTS

### PRECONDITIONS

> Pour qu'un voiture puisse s'approcher, il faut qu'elle soit loin du carrefour

*Approche : Position = Loin_avant*

> Pour qu'une voiture puisse entrer dans le carrefour, il faut que les conditions suivantes soient remplies :
>
> 1. elle est devant le carrefour
> 2. elle est la suivante à entrer
> 3. le feu n'est pas rouge

*Entre : Position = Devant_feu*
*Λ Intention.Bande_origine.Voit_suiv = SELF*
*Λ ¬(Intention.Feu_regie.Couleur = Rouge)*

> Pour qu'une voiture puisse sortir du carrefour, il faut qu'elle remplisse les condition suivantes :
>
> 1. Elle est dans le carrefour
> 2. Elle est la suivante à sortir

*Sort : Position = Dans_carref*
*Λ Intention.Voiture_suiv = SELF*

### EFFECT OF ACTIONS

*Approche : [] Position := Devant_feu*

*Entre : [] Position := Dans_carref*

*Sort : [] Position := Loin_apres*

## COOPERATION CONSTRAINTS

### STATE PERCEPTION

```
La voiture voit la couleur du feu relatif a sa bande si elle est juste devant.
```

*K(Feu.Couleur / b = Intention.bande_origine Λ b.Voit_suiv = SELF)*

```
La voiture voit si elle est la suivante a pouvoir entrer dans le carrefour
```

*K(b.Voit_suiv / b = Intention.bande_origine Λ b.Voit_suiv = SELF)*

```
La voiture voit sur quelle bande elle se trouve
```

*K(traj.Bande_origine / Intention = traj)*

```
La voiture ne peut voir que le feu qui regle la trajectoire qu'elle veut
emprunter
```

*K(traj.Feu_regie / Intention = traj)*

```
La voiture voit si elle est la suivante a pouvoir sortir du carrefour
```

*K(traj.Voiture_suiv / Intention = traj Λ traj.Voiture_suiv = SELF)*

```
La voiture voit sur quelles trajectoires peuvent se trouver des voitures la
genant. Elle ne les voit que si elle est la suivante a pouvoir sortir du carrefour
```

*K(traj.Traj_ralentissantes / Intention = traj Λ traj.Voiture_suiv = SELF)*

```
La voiture voit les voitures qui la genent.
```

*K(traj.Voitures / traj ∈ Intention.Traj_ralentissantes Λ Intention.Voiture_suiv = SELF)*

### ACTION INFORMATION

*K(Approche.Bande / TRUE)*

*K(Entre.Bande / TRUE)*

*K(Entre.Trajectoire / TRUE)*

*K(Sort.Trajectoire / TRUE)*

## AGENT CARREFOUR.FEU

## DECLARATION

> Represente la classe des feux dans le carrefour

### STATE COMPONENTS

> Retient la couleur actuelle du feu

*Couleur* instance-of *COULEUR* → *Carrefour.Voiture Carrefour.Feu*

> Un element de cet ensemble represente une instance d'un autre feu et une couleur.
>
> Si le couple (f, c) fait partie de cet ensemble, alors le feu actuel est synchronis avec le feu f pour la couleur c.
>
> Un feu a est synchronise avec un feu b sur la couleur c
>
> ssi (b a la couleur c => a a la couleur c)

**Synchronise* set-of *SYNCHRO_FEU*

> Donne des contraintes d'exclusions sur les feux.
>
> Si f est un element de cet ensemble pour le feu b
>
> Alors (f a la couleur verte) => not(b a la couleur verte)

**Exlusion* set-of set-of*Carrefour.Feu*

**Temps_orange* instance-of *DURATION*

**Temps_max_rouge* instance-of *DURATION*

**Temps_min_rouge* instance-of *DURATION*

**Temps_max_vert* instance-of *DURATION*

**Temps_min_vert* instance-of *DURATION*

> Ce composant d'etat indique la priorite de ce feu par rapport aux autres feux.
>
> Si beaucoup de voitures attendent devant ce feu, alors sa priorite est grande.

*Priorite* instance-of *RATIONAL* → *Carrefour.Feu*

### ACTIONS

**Passage_vert*

**Passage_orange*

**Passage_rouge*

> Cette action sert uniquement a modeliser le changement de priorite qui est entierement regle par des contraintes de type state behavior

**Chmnt_priorite(RATIONAL)*

## DECLARATIVE CONSTRAINTS

### STATE BEHAVIOR

La traduction de la signification de la synchronisation des feux.

$[]\ (Couleur = x) \leq\!> \forall fs : (fs \in Synchronise \wedge fs.Coul = x \wedge fs.Feu.Couleur = x)$

La traduction de la signification d'exclusion de feux.

$[]\ \exists f : (f \in Exclusion \wedge f.Couleur = Vert) => Couleur <> Vert$

Cette contrainte ensemble avec la suivante permet de definir la priorite du feu comme le maximum des priorites des bandes concernes par ce feu.

$[]\ \forall b : (b \in Bande \wedge b.Priorite \leq Priorite)$

Cette contrainte ensemble avec la precedente permet de definir la priorite du feu comme le maximum des priorites des bandes concernes par ce feu.

$[]\ \exists b : (b \in Bande \wedge b.Priorite = Priorite)$

Lorsque un carrefour des environs est bloque, il faut fermer toutes les trajectoires menant vers ce carrefour. Cela se fait en mettant sur rouge tous les feux regissant ces trajectoires.

$[]\ (\exists t : t \in Trajectoire \wedge t.Aboutissement.Bloque) => Couleur = Rouge$

## OPERATIONAL CONSTRAINTS

### PRECONDITIONS

Cette contrainte assure que le feu reste au minimum un certain temps au rouge

$Passage\_vert : Couleur = Rouge$ Since $Temps\_min\_rouge$

Cette contrainte assure que le feu reste au minimum un certain temps à l'orange

$Passage\_orange : Couleur = Vert$ Since $Temps\_min\_vert$

Cette contrainte assure que le feu reste au minimum un certain temps au vert

$Passage\_rouge : Couleur = Vert$ Since $Temps\_orange$

### EFFECT OF ACTIONS

$Passage\_vert : [] Couleur := Vert$

$Passage\_orange : [] Couleur := Orange$

$Passage\_rouge : [] Couleur := Rouge$

$Chmnt\_priorite(p) : [] Priorite := p$

### TRIGGERINGS

| Cette contrainte fait que le feu ne reste qu'un certain temp maximum au rouge

$Couleur = Rouge / Temps\_max\_rouge \rightarrow Passage\_vert$

| Cette contrainte fait que le feu ne reste qu'un certain temp maximum au vert

$Couleur = Vert / Temps\_max\_vert \rightarrow Passage\_orange$

| Cette contrainte fait que le feu ne reste qu'un certain temp maximum à l'orange

$Couleur = Orange / Temps\_orange \rightarrow Passage\_rouge$

| Si le feu actuel est vert et que tous les feux ayant une contrainte d'exclusion ont une priorite plus grande pour passer au vert, alors le feu doit devenir rouge.

$Couleur = Vert \Lambda (\forall f : (f \in Exclusion) \Lambda (f.Priorite > Priorite)) / 1sec \rightarrow Passage\_orange$

## COOPERATION CONSTRAINTS

### STATE PERCEPTION

| Vue que la modelisation met "l'intelligence" du systeme au niveau des feux, ceux-ci sont au courant des couleurs des autres feux.

$K(Feu.Couleur / TRUE)$

| Un feu ne voit que les priorites des bandes qui sont concernes par ce feu

$XK(b.Priorite / \exists t : t.Bande\_origine = b \Lambda t.Feu\_regie = SELF)$

| Le feu ne voit que les carrefours des environs vers lesquels mene une trajectoire reglee par ce feu.

$XK(c.Bloque / \exists t : t.Aboutissement = c \Lambda t.Feu\_regie = SELF)$

| Un feu voit vers quel carrefour de l'environ menent les trajectoires qu'il regle

$XK(t.Aboutissement / t.Feu\_regie = SELF)$

| Le feu ne se coordonne sur la priorite qu'avec les feux ayant une contrainte d'exclusion avec lui-meme

$XK(f.Priorite / f \in Exclusion)$

### STATE INFORMATION

$K(Couleur.Voiture / TRUE)$

$K(Couleur.Feu / TRUE)$

$K(Priorite.Feu / TRUE)$

## AGENT CARREFOUR.BANDE

## DECLARATION

> Represente les voies d'approche au carrefour.
>
> Deux voitures ne peuvent etre de front sur une meme bande.
>
> La bande retient la file des voitures se trouvant sur elle et traite les informations recu du systeme de detection de presence de voitures.

### STATE COMPONENTS

> La sequence des voitures presentes sur cette bande.
>
> Il s'agit d'une file d'attente devant le feu

*Voitures \sequence-of \sequence-ofCarrefour.Voiture*

> La voiture directement devant le feu rouge

*Voit_suiv* instance-of instance-of*Carrefour.Voiture* derived-from *Voitures → Carrefour.Voiture*

> La capacite maximale en nombre de voitures visibles du systeme de detection de presence de voitures

**Capacite_SDPV* instance-of *INTEGER*

> Le nombre de voitures detectes effectivement par le systeme

*Nbr_voit_detectes* instance-of *INTEGER* derived-from *Capacite_SDPV Voitures → Carrefour.Feu*

> La priorite est d'autant plus grande qu'il y ait de voitures en attente dans la bande

*Priorite* instance-of *RATIONAL* derived-from *Capacite_SDPV Nbr_voit_detectes*

## BASIC CONSTRAINTS

### DERIVED COMPONENTS

*Voit_suiv ≙ First(Voitures)*

*Nbr_voit_detectes ≙ Min(Capacite_SDPV, Count(Voitures))*

> La priorite est calcule en fonction du nombre de voitures detectees. Si le nombre maximal detectable est atteint, alors la priorite est maximale.
>
> Vu que la capacite peut etre nulle et que la division par zero est une operation non definie, on divise par 1 au minimum.

*Priorite ≙ Nbr_voit_detectes / Max(Capacite_SDPV, 1)*

### INITIAL VALUATION

> Au debut, il n'y a pas de voitures sur les bandes

*Voitures = []*

## OPERATIONAL CONSTRAINTS

### EFFECT OF ACTIONS

*voit.Approche : [] Voitures := Append(Voitures, voit)*

*voit.Entre : [] Voitures := Remove(Voitures, voit)*

## COOPERATION CONSTRAINTS

### ACTION PERCEPTION

La bande ne voit approcher que les voitures arrivant sur elle-meme.

*XK (voit.Approche / voit.Intention = traj Λ traj.Bande_origine = SELF)*

La bande ne voit entrer les voitures dans le carrefour que si elles viennent d'elle-meme.

*XK (voit.Entre / voit.Intention = traj Λ traj.Bande_origine = SELF)*

### STATE INFORMATION

*XK(Voit_suiv.v / v = Voit_suiv)*

*K(Priorite.Feu / TRUE)*

# AGENT CARREFOUR.TRAJECTOIRE

## DECLARATION

Represente un chemin que peut parcourir une voiture dans le carrefour.

L'agent trajectoire retient les voitures qui s'y trouvent et connait ses connections avec les autres agents :

la bande donnant sur la trajectoire,

le feu qui l'ouvre ou la ferme,

les trajectoires qui la coupent (trajectoires ralentissantes)

le carrefour de l'environnement sur lequel elle debouche

Elle ne peut contenir qu'un nombre maximal de voitures.

### STATE COMPONENTS

Les voitures engages dans le carrefour sur cette trajectoire.

Il s'agit d'une file d'attente.

*Voitures \sequence-of \sequence-ofCarrefour.Voiture → Carrefour.Voiture*

Les voitures empruntant cette trajectoire viennent de cette bande.

*Bande_origine* instance-of instance-of*Carrefour.Bande* → *Carrefour.Voiture*

| Le nombre maximal de voitures pouvant etre engages dans le carrefour sur cette trajectoire

*Nb_max_voit* instance-of *INTEGER*

| Le feu determinant si une voiture peut passer de la bande dans cette trajectoire

*Feu_regie* instance-of instance-of*Carrefour.Feu* → *Carrefour.Voiture*

| Ensemble des trajectoires ralentissantes.
|
| Une trajectoire A est ralentissante par rapport à une trajectoire B si la presence d'une voiture dans la trajectoire A ralentit fortement le nombre de voitures par unite de temps pouvant quitter le carrefour via la trajectoire B.

*Traj_ralentissantes* set-of set-of*Carrefour.Trajectoire* → *Carrefour.Voiture*

| La voiture suivante pouvant sortir du carrefour

*Voiture_suiv* instance-of instance-of*Carrefour.Voiture* derived-from *Voitures* → *Carrefour.Voiture*

| Donne le carrefour des environs vers lequel les voitures vont en sortant du carrefour par cette trajectoire

*Aboutissement* instance-of instance-of*Carrefour.Autre_carrefours* → *Carrefour.Feu*

## BASIC CONSTRAINTS

### DERIVED COMPONENTS

*Voiture_suiv* ≙ *First(Voitures)*

### INITIAL VALUATION

| Au depart, il n'y a aucune voiture dans le carrefour

*Voitures = []*

## DECLARATIVE CONSTRAINTS

### STATE BEHAVIOR

| La trajectoire a une capacite maximale qui ne peut etre depassee

*[] Count(Voitures)* ≤ *Nb_max_voit*

## OPERATIONAL CONSTRAINTS

### EFFECT OF ACTIONS

*voit.Entre : [] Voitures := Append(Voitures, voit)*

*voit.Sort : [] Voitures := Remove(Voitures, voit)*

## COOPERATION CONSTRAINTS

### ACTION PERCEPTION

```
La trajectoire voit les voitures qui entrent arrivnt sur elle.
```

*XK (voit.Entre / voit.Intention = SELF)*

```
La trajectoire voit les voitures qui la quittent
```

*XK(voit.Sort / voit.Intention = SELF)*

### STATE PERCEPTION

*XK(Voiture.Intention / TRUE)*

### STATE INFORMATION

*K(Bande_origine.Voiture / TRUE)*

*K(Feu_regie.Voiture / TRUE)*

*XK(Voiture_suiv.v / v = Voiture_suiv)*

*K(Traj_ralentissantes.Voiture / TRUE)*

*K(Voitures.Voiture / TRUE)*

*K(Aboutissement.Feu / TRUE)*

# AGENT CARREFOUR.AUTRE_CARREFOURS

## DECLARATION

```
Cet agent represente un autre carrefour duquel on sait juste s'il est bouche ou
non.
```

### STATE COMPONENTS

```
Indique si le carrefour est bloque ou non
```

*Bloque* instance-of *BOOLEAN → Carrefour.Feu*

### ACTIONS

**Bloquage*

**Debloquage*

## BASIC CONSTRAINTS

### INITIAL VALUATION

*Bloque = FALSE*

## OPERATIONAL CONSTRAINTS

### EFFECT OF ACTIONS

*Bloquage : [] Bloque := TRUE*

*Debloquage : [] Bloque := FALSE*

## COOPERATION CONSTRAINTS

### STATE INFORMATION

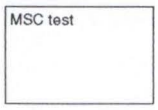
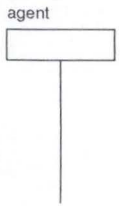
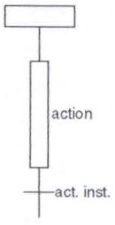
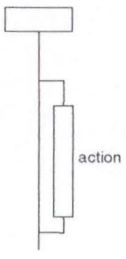
*K(Bloque.Feu / TRUE)*

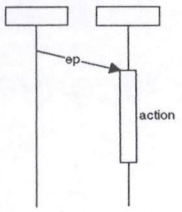
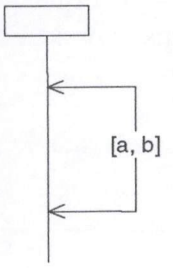
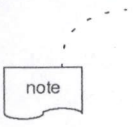
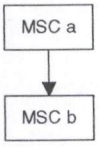
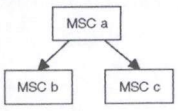
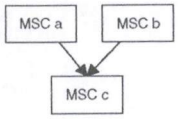
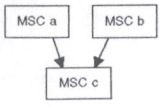
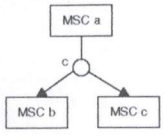
## 7.2 Annexe 2 – Syntaxe et sémantique de base du langage

### 7.2.1 Contenu

Le tableau suivant reprend les constructions syntaxiques du langage de représentation du contenu d'un scénario. Ce langage est basé sur les diagrammes MSC munis de quelques extensions. Il existe deux types de schémas : les bMSC et les hMSC. La ligne 'application' indique à quel type de schéma la construction syntaxique s'applique.

A la fin du tableau, on trouve un exemple reprenant toutes les constructions syntaxiques sur un même schéma. Il y a un schéma de type bMSC et un de type hMSC.

| **Construction syntaxique** | **Explications** |
|---|---|
|  | *Application* : bMSC et hMSC<br>Représentation d'un épisode. L'épisode est identifié par le nom se trouvant en haut à gauche du rectangle. Les détails de l'épisode peuvent être dessinés à l'intérieur du rectangle. |
|  | *Application* : bMSC<br>Représentation d'une partie d'une vie d'un agent. Le nom de l'instance de l'agent se trouve au-dessus du rectangle supérieur. La ligne verticale vers le bas représente la ligne du temps débutant au rectangle.<br>Des vies de plusieurs agents peuvent être représentées l'une à côté de l'autre. Des hauteurs égales sur les lignes du temps de ces vies représentent des mêmes moments dans le temps. |
|  | *Application* : bMSC<br>Représentation d'une occurrence d'une action. L'élargissement de la ligne du temps représente le temps pendant lequel une certaine action se déroule. Le nom de l'action est marqué à côté de l'élargissement. Une action instantanée est représentée par une petite barre verticale |
|  | *Application* : bMSC<br>Représentation d'une déviation de la ligne du temps. Une déviation peut être introduite à n'importe quel endroit d'une ligne du temps et sert en général pour représenter des actions se déroulant en parallèle. Toutes les constructions syntaxiques attachées à une ligne du temps normale peuvent être appliquées à une déviation de la ligne du temps. |

| | |
|---|---|
|  | *Application* : bMSC<br>Représentation de la cause d'une occurrence d'action. Une flèche arrivant au début de la représentation d'une action représente une cause de l'occurrence de cette action. La cause est spécifiée par le libellé de la flèche. Une telle flèche ne peut arriver qu'au début d'un élargissement de la ligne du temps.<br>Le début de la flèche représente le moment de la perception de la cause. |
|  | *Application* : bMSC et hMSC<br>Représentation d'une contrainte de temps. Les deux pointes de la flèche représentent les moments sur la ligne du temps sur lesquels la contrainte s'applique. La contrainte est décrite par le libellé de la flèche et a la forme [a, b]. Elle signifie que le temps s'écoulant entre les deux pointes de la flèche prend entre a et b unités de temps.<br>Dans le cas des hMSC, les pointes arrivent à des flèches liant des épisodes entre eux. |
|  | *Application* : bMSC et hMSC<br>Représentation d'une note informelle. Le contenu du 'rectangle' est le texte de la note. Une note peut être attachée à un élément particulier du diagramme grâce à une ligne pointillée. |
|  | *Application* : hMSC<br>Représentation d'un enchaînement séquentiel. Le MSC b débute lorsque le MSC a est terminé. |
|  | *Application* : hMSC<br>Représentation d'un enchaînement parallèle. Les MSC b et MSC c débutent lorsque le MSC a est terminé. |
|  | *Application* : hMSC<br>Représentation d'un enchaînement séquentiel avec synchronisation. Deux flèches arrivant au même point sur un épisode représentent un enchaînement séquentiel avec synchronisation. Le MSC c débute lorsque le MSC a *et* le MSC b sont terminés. |
|  | *Application* : hMSC<br>Représentation d'un enchaînement séquentiel avec duplication. Deux flèches arrivant à deux points différents sur un même épisode représentent un enchaînement séquentiel avec duplication. Deux instances de l'épisode MSC c existent, l'un débutant lorsque le MSC a est terminé et l'autre lorsque le MSC b est terminé. |
|  | *Application* : hMSC<br>Représentation d'un point de décision. Un petit cercle représente un point de décision. La condition est représentée à côté du rond et prend la syntaxe d'une expression de la logique des prédicats du premier ordre. Elle peut utiliser des méta-variables. |

| | |
|---|---|
| ▽ △ | *Application* : hMSC<br>Représentation des points de début et de fin du hMSC. Un triangle 'renversé' (celui de gauche) représente le début de l'épisode. Il n'y en a qu'un. Un triangle 'normal' représente un point de sortie du scénario. Il peut y en avoir plusieurs |

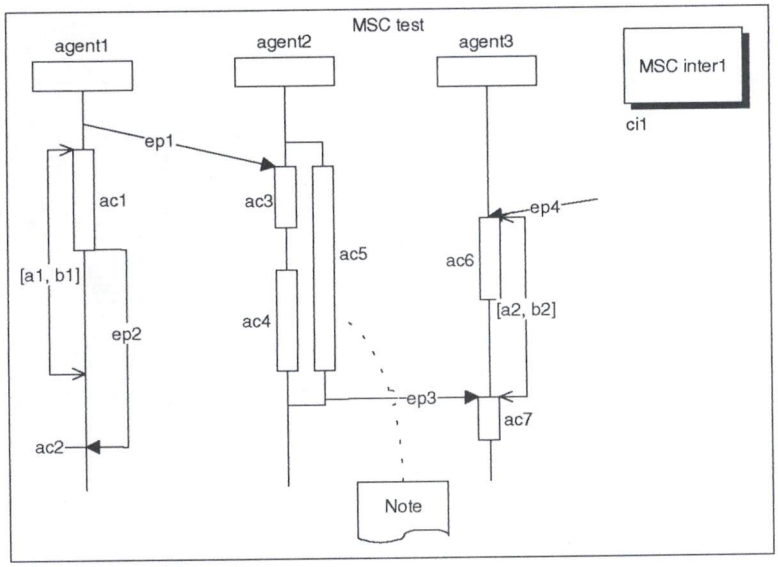



**Figure 39 - toutes les constructions syntaxiques d'un bMSC**

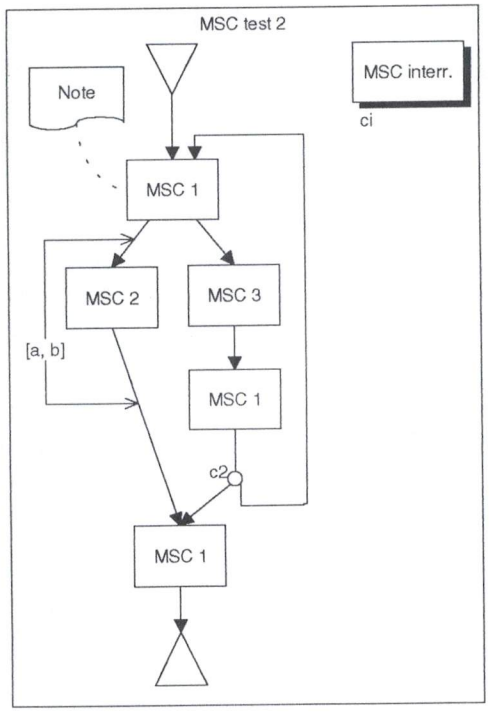



**Figure 40 - toutes les contructions syntaxiques d'un hMSC**

### 7.2.2 Contexte formel

Le tableau suivant reprend les variables pouvant être utilisées dans le cadre d'une précondition ou postcondition d'un scénario ALBERT. Il reprend également les constructions syntaxiques utilisées pour la diminution des indéterminations rencontrés dans une spécification ALBERT et cela dans le but d'une animation basée sur le scénario concerné.

| **Variable ou construction syntaxique** | **Explication** |
|---|---|
| *Base* | |
| `Inst_agent` | type : `table[STRING -> set{STRING}]`<br>`Inst_agent['voiture']` dénote l'ensemble des instances de l'agent voiture |
| `State_comp` | type : `table[(STRING, STRING, STRING) - DATATYPE]`<br>`State_comp[('voiture', 'Plaque 1Z483', 'position')]` dénote la valeur du composant d'état 'position' de l'instance 'Plaque 1Z483' de l'agent 'voiture'. |
| `Act_occ` | type : `table[(STRING, STRING, STRING) -> seq{(INTEGER, REAL, REAL)}]`<br>`Act_occ['voiture', 'Plaque 1z483', 'entre_carrefour']` dénote la séquence des couples déterminant le début et la fin des instances de l'action 'entre_carrefour' de l'instance 'Plaque 1Z483' de l'agent 'voiture'. La valeur entière associée au couple du temps de début et de fin est utilisé pour identifier une instance précise. |
| `Perceptions` | type : `table[(STRING, STRING, STRING, INTEGER) -> set{STRING, STRING)]`<br>`Perceptions[('voiture', 'Plaque 1Z483', 'entre_carrefour', 1)]` dénote l'ensemble des noms et instances d'agents percevant l'occurrence n° 1 de l'action 'entre_carrefour' de l'instance 'Plaque 1Z483' de l'agent 'voiture'. |
| *Diminuer l'indétermination* | |
| Nbr d'acteurs | Fixer le nombre d'acteurs peut se faire grâce à une contrainte du style :<br>`count['nom agent'] = nbr d'acteurs` |
| Valeur initiale d'un composant d'état | La valeur initiale d'un composant d'état peut être fixée grâce à une contrainte du style :<br>`State_comp['nom agent', 'nom instance' ou variable, 'nom comp. d'état'] = valeur` |
| Probabilités de perception d'actions | Les probabilités de perception des actions peuvent être fixées grâce à des contraintes portant sur la variable suivante : `proba_perc`. Elle est définie de la manière suivante.<br>Type : `table[(STRING, STRING, STRING) -> set{(STRING, STRING, INTEGER)}]`<br>`Proba_perc[x, y, z] = (a, b, c)` ⇔ |

| | |
|---|---|
| | `c = n / m ∧`<br>`n = Count(act_occ[x, y, z]) ∧`<br>`m = Count(Y) ∧`<br>`(a, b, t) ∈ Y ∧`<br>`(a, b) ∈ perceptions[x, y, z, t]` |
| Durée d'action | Les durées des actions peuvent être limitées ou fixées grâce à des contraintes faisant référence à la variable suivante : `Act_dur`. Elle est définie de la manière suivante.<br>Type : `table[(STRING, STRING, STRING, INTEGER) -> REAL]`<br>`Act_dur[a, b, c, d] = z ⇔`<br>`z = x - y ∧`<br>`(d, x, y) ∈ act_occ[a, b, x]` |
| Début d'action | Le moment du début d'une action peut être fixé grâce à une contrainte du style suivant :<br>`Act_occ['nom agent', 'nom instance', 'nom action'] = (numéro instance, début action, fin action)` |

### 7.2.3 Contexte informel

Le tableau suivant reprend les différentes rubriques utilisées pour retenir des informations relatives à la construction d'un scénario dans un but d'amélioration de la gestion de l'ensemble des scénarios. Il existe 3 types de rubriques. Les noms des types de rubriques sont écrits en *italique*.

Le tableau ci-dessous résume le modèle complet que nous avons défini.

| **Nom rubrique** | **Description** |
|---|---|
| *Spécification du scénario* | |
| Nom | Identifiant de l'épisode ou du scénario. |
| Brève description | Deux ou trois lignes décrivant de manière informelle le scénario. |
| But | Objectif du scénario. Cela comporte d'une part le pourquoi de ce scénario et d'autre part les objectifs de certains agents intervenant dans le scénario. |
| Mots clés | |
| Service rendu | Un scénario assez élaboré faisant intervenir un utilisateur du futur système décrit en général un service que doit rendre le système à cet utilisateur. Cette rubrique permet de le décrire. |
| Agents intervenants | Enumération des agents intervenants dans le scénario ou l'épisode. |
| *Informations de création* | |
| Fournisseurs d'information | L'ensemble des personnes ayant apporté l'information nécessaire à la création du scénario ou de l'épisode. Pour chaque personne, on connaît son nom et un moyen de la contacter facilement. |
| Analyste | L'ensemble des personnes ayant formalisé et écrit les scénarios. Pour chaque |

| | |
|---|---|
| | personne on connaît son nom et un moyen de la contacter. |
| Date de dernière modification | La date de la dernière modification ayant été apportée au scénario ou à l'épisode. |
| Version | Une indication sur la version du scénario. Cette indication est nécessaire pour les traces et doit également comporter la mention 'en cours d'élaboration' ou 'terminé'. |
| Trace | Un ensemble de noms de scénarios ou d'épisodes ainsi que les versions de ceux-ci permet d'indiquer les étapes parcourues pour arriver à ce scénario ou à cet épisode et de retrouver facilement les produits intermédiaires. |
| *Documentation supplémentaire* | |
| Dessin de l'environnement | Le ou les dessins facilitant la compréhension du scénario souvent en explicitant l'environnement. |
| Références | Toutes les références bibliographiques des ouvrages consultés pour comprendre et élaborer le scénario ou épisode. |
| Spécification ALBERT | Une référence vers la spécification ALBERT définissant le vocabulaire relatif à ce scénario. Cette référence doit aussi comprendre la version de la spécification. |
| Notes | Des notes supplémentaires intéressantes pour la compréhension du scénario ou de l'épisode. |