THESE

présentée à

L'UNIVERSITE DES SCIENCES ET TECHNOLOGIES DE LILLE
en vue de l'obtention du grade de

DOCTEUR DE L'UNIVERSITE

en

PRODUCTIQUE,
AUTOMATIQUE ET INFORMATIQUE INDUSTRIELLE

par

Adam BOURAS
Ingénieur

CONTRIBUTION A LA CONCEPTION D'ARCHITECTURES REPARTIES : MODELES GENERIQUES ET INTEROPERABILITE D'INSTRUMENTS INTELLIGENTS.

Soutenue publiquement le 02 Juillet 1997 devant la commission d'examen

Membres du jury :

rapporteurs:	Professeur **Laurent FOULLOY.**
	Professeur **Jean-Jacques LESAGE.**
examinateurs:	Professeur **Jean-Louis FERRIER.**
	Professeur **Mireille BAYART-MERCHEZ.**
Directeur de thèse	Professeur **Marcel STAROSWIECKI.**
Invité	**Philippe DESODT.**

Thèse effectuée au LAIL URA CNRS 1440D

A mes parents

-AVANT PROPOS-

Ce mémoire est le résultats de travaux menés au Laboratoire d'Automatique et d'Informatique industriel de Lille (URA CNRS 1440D) sous la direction de Monsieur le Professeur Marcel STAROSWIEKI professeur à l'Ecole Universitaire d'Ingénieur de Lille dont je tiens à exprimer ma plus profonde gratitude et dont les conseils et la rigueur scientifique sont pour l'essentiel dans les résultats de ce travail.

Je tiens aussi à exprimer mes vifs remerciements aux différents membres du jury qui me font l'honneur de participer à l'examen de ce travail:

Monsieur le professeur Jean-Jacques LESAGE Professeur des universités à l'Ecole Supérieur de CACHAN.

Monsieur le professeur Laurent FOULLOY Professeur à l'Ecole Supérieur d'ingénieur d'ANNECY, université de savoie.

Monsieur le professeur Jean Louis FERRIER Professeur à l'université d'ANGERS.

Madame le professeur Mireille BAYART-MERCHEZ Professeur à l'Ecole Universitaire d'Ingénieur de Lille.

Mes remerciements vont aussi à la personne de Monsieur Philippe DESODT directeur du Laboratoire d'Electronique de l'Ecole des Mines de DOUAI pour l'accueil et le soutien financier apporté, par l'Ecole des Mines, pour la réalisation de ce travail.

RESUME

L'expression "instruments intelligents" est apparue à partir du moment où les instruments de terrain mis en œuvre dans les systèmes automatisés (capteurs et actionneurs) ont disposé de leur propre puissance de traitement grâce au développement des technologies permettant la réalisation des microprocesseurs et des micro-contrôleurs. Toutefois, les travaux effectués jusqu'à présent sur le thème de l'instrumentation intelligente se sont toujours intéressés à la nature des structures et des traitements implantés dans un instrument. Ces travaux s'adressent donc plus particulièrement aux concepteurs. Les utilisateurs-intégrateurs d'architectures réparties avec instruments intelligents manifestent d'autres besoins et se préoccupent généralement peu de la nature des structures internes. Il convient donc de formaliser une approche externe (ou utilisateur) dans le concept de l'instrumentation intelligente. Dans ce mémoire nous avons cherché à faire reposer la conception d'architectures réparties comportant des capteurs et actionneurs intelligents sur une démarche formelle.

Pour cela, il convient tout d'abord de définir un modèle générique externe d'instrument intelligent, instancié pour chaque instrument particulier. Ce modèle doit pouvoir être composé, pour construire celui d'un ensemble d'instruments interconnectés. La réalisation d'une application donnée, par un tel ensemble d'instruments, se traduit par la mise en place de mécanismes traduisant ses contraintes spécifiques. Nous avons développé une démarche, fondée sur la notion de menu, pour spécifier les règles de conduite qui en découlent. Enfin, à un niveau hiérarchique plus élevé, nous proposons une méthode pour construire l'ensemble des séquences de conduite automatisables d'une application donnée prise en charge par un ensemble d'instruments interconnectés.

ABSTRACT

The expression "intelligent instrument" has appeared when field instruments in automated systems (sensor and actuators) were provided with their own processing power, as a result of micro-electronic technology development. However, research made on the intelligent instrument concept has been always interested on the nature of the structures of the implemented treatments on the instrument. The uses-integrators of distributed architectures with intelligent instruments exhibit others needs and are generally no interested with the nature of the internal structures. It is then convenient to formalize an external approach in the intelligent instrument concept. In this research we tried to make be built the intelligent instrument concept on a formal approach.

For this, we define an external generic descriptive model for intelligent instrument. This model can be composed to construct the model of a set of interconnected instruments. The realization of a given application by a such set of instruments is translated by the setting up of a mechanism to express its specific constraints. We have developed an approach based on the menu notion, to specify the operating rules that follows from the specific constraints of an application. Finally, at higher hierarchical level, we propose a method to build the set of all the automatable operating sequences for a given applicati built by interconnecting a set of intelligent instruments.

Table des figures

Sommaire

Première partie
Modèle générique pour l'intéropérabilité d'instruments intelligents

Chapitre 1. Introduction générale

1.1. Synthèse et historique.

Dès le début des années 80, l'automatisation de plus en plus complexe des systèmes de production, a fait émerger des capteurs et actionneurs possédant de larges possibilités de configuration et de communication. Ces équipements sont souvent caractérisés par le fait qu'ils se présentent comme des "boites noires" avec lesquelles il est possible de communiquer. Etant donné le foisonnement des possibilités de traitement offertes, du nombre d'implantations possibles et de la diversité des origines de ce type d'équipement [NAJA 92], il est impératif que soient définies des règles de conception permettant, dans le cadre d'un système automatisé à architecture distribuée, une compatibilité entre les différents équipements offerts par les différents constructeurs.

Les premières tentatives dans le but de définir des règles générales pour l'interconnexion d'instruments intelligents sont les travaux de normalisations des différents bus de terrain. La dernière normalisation en date est la norme Européenne EN50170. Les supports théoriques de ces différentes normalisations sont les travaux de [ZIMM 80] structurant en sept couches OSI (Open System Interconnection) les problèmes à résoudre pour l'établissement d'une communication entre plusieurs équipements. Certains de ces travaux de normalisation aboutissent à des définitions de l'interopérabilité et au développement de langages de description [WEST 92]. Cependant, ces travaux ne tiennent pas compte de l'application réalisée ; de plus, les résultats obtenus restent totalement dépendants du support de communication considéré.

Par ailleurs, il est évident que la conception d'applications distribuées à partir d'instruments intelligents issus de différents constructeurs n'est pas uniquement basée sur l'utilisation d'un support de communication standard. Pour cette raison, les premières réflexions sur le thème

de l'instrumentation intelligente sont apparues avec le [CIAME] et concernent l'étude, indépendante du support de communication, des fonctions nécessaires à un instrument intelligent pour garantir une interconnexion facile dans le cadre d'une automatisation à architecture distribuée. Ces travaux ont conduit à l'élaboration du modèle interne d'un instrument intelligent [GEHI 94]. Ce modèle (à l'image du modèle OSI) définit les différents blocs fonctionnels que doit présenter un instrument intelligent et leurs interconnexions afin de garantir sa connexion dans le cadre d'une automatisation distribuée.

Toutefois, les travaux effectués jusqu'à présent sur le thème de l'instrumentation intelligente se sont toujours intéressés à la nature des structures et des traitements implantés dans un instrument. Ces travaux s'adressent donc plus particulièrement aux concepteurs. Les utilisateurs, quant à eux, se préoccupent peu de connaître les structures internes d'un instrument intelligent. Ce qu'ils veulent avant tout c'est surtout savoir ce qu'ils peuvent obtenir d'un instrument ? comment s'y prendre pour y parvenir ? quelles sont les conduites autorisées par une architecture répartie interconnectant des instruments intelligents ? quelle conduite adopter pour réaliser une application particulière ? dans quelles conditions ceci est-il possible ? etc. Toutes ces questions se posent tout naturellement pour l'ensemble des instruments intelligents sans distinction, il est donc évident qu'il ne peut y avoir une réponse par capteur et par actionneur intelligent. Pour répondre à ces questions il est indispensable de définir une approche générale indépendante des structures internes et qui permet de modéliser la vision que peut avoir l'utilisateur. Il existe donc un besoin de formaliser une approche externe (ou utilisateur) dans le concept de l'instrumentation intelligente.

1.2. Le concept d'instrumentation intelligente.

L'expression "instruments intelligents" est apparue à partir du moment où les instruments de terrain mis en œuvre dans les systèmes automatisés (capteurs et actionneurs) ont disposé de leur propre puissance de traitement grâce au développement des technologies permettant la réalisation des microprocesseurs et des micro-contrôleurs. Toutefois, le terme "intelligent" ne doit pas prêter à confusion. Cet adjectif qualifie ici des objets finalisés dont la vocation initiale, grâce aux possibilités de traitement offertes par la micro-électronique, s'est élargie jusqu'à la participation à un certain nombre de fonctionnalités relevant à l'origine du système

d'automatisation dans son ensemble. Dans ce sens les termes anglais "Smart sensors" et "Smart actuators" sont plus appropriés.

Dans ce contexte, le concept d'instrumentation intelligente englobe la définition des différentes notions et critères permettant, dans le cadre d'une automatisation intégrée, la prise en compte de l'impact de la révolution technique dans le domaine des capteurs et actionneurs. Il s'agit donc d'un concept ouvert permettant de concevoir et/ou de prévoir le fonctionnement global d'applications construites à partir d'instruments issus de diverses origines et présentant des potentialités techniques variées. Dans le cadre de ce concept, nous nous intéressons plus particulièrement aux problèmes liés à la description des fonctionnalités offertes par un instrument intelligent ainsi qu'à ceux liés à la réalisation d'une application donnée à partir d'un ensemble d'instruments interconnectés.

En effet, l'impact de la révolution technique dans le domaine des capteurs et actionneurs peut être étudié en analysant ce type d'équipement sous trois points de vue [STAR 96 a] :

- Comme outils dédiés à une fonction précise, les performances de chacun dépendent de la puissance des traitements implantés.
- Comme produits industriels pouvant passer durant leur cycle de vie par plusieurs phases différentes. La phase d'exploitation n'est pas la seule phase où l'ont peut bénéficier des services offerts par un instrument intelligent. Dans les phases de configuration, d'implantation, de test ou de maintenance un capteur ou actionneur intelligent peut proposer différents services à ses différents utilisateurs.
- Comme composants d'un système automatisé, ils contribuent à la réalisation de la fonction globale du système. En effet, les systèmes automatisés deviennent de plus en plus complexes et ont besoin de traiter de plus en plus d'informations pour respecter les contraintes auxquelles ils sont soumis. Ainsi, les architectures classiques ne peuvent pas facilement prendre en compte le résultat de cette croissance en flux de données à traiter, elles sont obligées d'évoluer vers des architectures distribuées. Au niveau du terrain, les capteurs et actionneurs intelligents sont les équipements les plus appropriés pour recevoir une partie de "l'intelligence" afin de réaliser localement une partie de la fonction globale du système automatisé.

Ces trois points de vue d'analyse conduisent à trois types d'approches possibles pour l'étude d'un instrument intelligent.

- Le premier point de vue conduit à une analyse descriptive de l'équipement en termes de potentialités techniques et de services pouvant être rendus à un utilisateur,
- Le second point de vue conduit à une analyse en termes de modes d'utilisation afin de spécifier les différents types d'exploitation possibles,
- Le troisième point de vue conduit à une analyse en termes de contraintes devant être respectées pour garantir l'interopérabilité d'un ensemble d'équipements dans le cadre d'une application bien définie.

Ce mémoire présente une contribution pour formaliser ces trois approches pour l'analyse d'un instrument intelligent.

1.3. Plan du mémoire:

Le présent mémoire est constitué de six chapitres qui sont organisés en trois parties.

Première partie
Modèle générique pour l'interopérabilité d'instruments intelligents.

La conception d'applications distribuées construites à partir d'instruments intelligents issus d'origines diverses nécessite une connaissance précise des fonctionnalités qu'offre chaque équipement. Il est donc indispensable de disposer d'une modélisation descriptive des potentialités techniques d'un instrument intelligent ainsi que de son langage de supervision. Pour cette raison, nous proposons dans le chapitre 2 un modèle générique qui sera instancié pour chaque instrument intelligent et permettra ainsi une spécification des fonctionnalités offertes.

Deuxième partie
Intéropérabilité d'instruments intelligents au sein d'une application répartie : spécification de la conduite.

La réalisation d'une application répartie par un ensemble d'équipements nécessite non seulement de disposer d'une modélisation rigoureuse de l'intelligence de chaque équipement, mais aussi d'exprimer la conduite à adopter pour réaliser la finalité de l'application. En effet la conduite d'un ensemble d'instruments intelligents pour la réalisation d'une application particulière est régie par des règles propres à l'application souhaitée. Pour cette raison, nous commençons tout d'abord dans le chapitre 3 par spécifier les différentes conduites possibles d'un ensemble d'instruments intelligents. Ensuite dans le chapitre 4 nous spécifions en termes de contraintes les règles de conduite à adopter pour réaliser l'application souhaitée. Dans le cas où ces règles de conduite sont autorisées par l'ensemble des instruments interconnectés, il est alors possible de prévoir leur interopérabilité, ceci bien sûr dans le cadre de l'application considérée.

Troisième partie
Automatisation de la conduite d'une application répartie construite en interconnectant des instruments intelligents.

A partir du moment où les règles de conduite que doit respecter l'utilisateur pour réaliser une application donnée par un ensemble d'instruments ont été spécifiées, il est envisageable de définir des niveaux de commande supérieurs permettant une conduite plus élaborée. L'objectif de cette partie est d'exprimer d'une manière formelle la démarche d'automatisation d'une application répartie interconnectant des instruments intelligents. Ainsi le chapitre 5 est dédié d'une part à la définition du procédé d'automatisation dans le contexte de l'instrumentation intelligente, d'autre part à la détermination des séquences de commandes qui peuvent être déléguées à un système d'automatisation pour la conduite d'une application donnée.

Le dernier chapitre de ce mémoire est un chapitre d'illustration par l'exemple des résultats obtenus. L'exemple choisi est un exemple académique, publié dans plusieurs articles différents. Les auteurs de l'exemple l'ont utilisé pour illustrer leur démarche d'automatisation par analyse décisionnelle. Dans ce dernier chapitre, nous spécifions de façon formelle la conduite ainsi que toutes les automatisations possibles du système étudié.

Chapitre 2. Modèle externe de spécification d'instruments intelligents

2.1. Introduction

L'émergence des capteurs et actionneurs intelligents pose le problème de la construction d'architectures de contrôle/commande distribuées interconnectant ces instruments au moyen de réseaux locaux de type bus de terrain [SIMO 95] [BAYA 95 a]. La conception de ces applications suppose leur décomposition en sous-applications prises en charge, pour chacune d'entre elles, par un instrument ou un sous ensemble d'instruments intelligents. Ainsi, un instrument intelligent réalise une partie de l'application globale en travaillant sur des variables produites localement (celles-ci constituent sa base de données locale) ou sur des variables produites par d'autres instruments (celles-ci lui seront communiquées par l'intermédiaire du réseau). Le fonctionnement global de tous les instruments d'une même application soulève des problèmes d'intéropérabilité. Ceux-ci apparaissent dès lors que :

- les données contenues dans les différentes bases de données locales doivent être partagées. Il est ainsi indispensable de spécifier les données présentes dans la base de donnée locale de chaque instrument intelligent. L'ensemble de toutes ces spécifications permettra alors à chaque instrument connecté au réseau de localiser les données utiles à son fonctionnement et de connaître la forme sous laquelle il pourra en disposer.
- les fonctions disponibles localement produisent des résultats utilisés par d'autres instruments. Cette mise à disposition de fonctionnalités propres à un instrument (à l'intention des autres instruments) nécessite que celles-ci, ainsi que le langage permettant d'y accéder à travers le réseau de communication, soient définies sans ambiguïté. La spécification, pour chaque instrument, des différentes

fonctionnalités offertes et de leur organisation permet de planifier la distribution des traitements d'une application distribuée [AKHA 96].

Ainsi, la prise en compte de l'aspect intéropérabilité dans le concept de l'instrumentation intelligente nécessite la description de l'équipement tel qu'il est vu à travers le réseau de communication c'est-à-dire d'un point de vue externe. Dans ces conditions, il importe peu de savoir comment les traitements internes sont réalisés : le problème est de savoir quelles sont les fonctionnalités d'un instrument intelligent dont nous pouvons bénéficier et comment y parvenir. La résolution du problème nécessite une démarche formelle de modélisation des instruments intelligents et de leur langage de commande. Le modèle proposé doit être un modèle générique à partir duquel il sera possible de décliner le modèle particulier propre à chaque instrument. Nous appelons ce modèle générique : modèle externe de spécification d'un instrument intelligent. L'élaboration de ce modèle fait l'objet du présent chapitre.

Pour un utilisateur externe, un instrument intelligent peut être considéré comme une entité proposant des services lesquels manipulent des variables et font appel à un ensemble de ressources. Ainsi, la notion de service est définie en adoptant une représentation de l'architecture matérielle de l'instrument intelligent identique à celle d'une machine informatique classique. Par ailleurs et afin d'éviter la réalisation par l'utilisateur d'actions incompatibles, les différents services d'un instrument sont regroupés en sous ensembles cohérents dits modes d'utilisation. Les conditions de passage d'un mode d'utilisation à l'autre peuvent être représentées dans un graphe état-transition.

Dans la dernière partie de ce chapitre, les différentes notions du modèle externe de spécification d'un instrument intelligent seront reprises puis structurées, tout d'abord suivant la norme Backus Naur pour la spécification des langages et ensuite dans une représentation arborescente permettant une vision globale. Ces deux structurations du modèle externe définissent des règles de spécification d'un instrument intelligent et constituent ainsi un support à une CAO du contrôle/commande d'applications distribuées interconnectant ces instruments au moyen de réseaux de communication.

2.2. Architecture matérielle d'un instrument intelligent

L'architecture matérielle d'un capteur ou d'un actionneur intelligent regroupe :

- des moyens de communication avec les opérateurs et/ou le système d'automatisation. Ces moyens permettent le transit, en entrée ou en sortie, des signaux et messages utilisés par les opérateurs ou les autres équipements de l'architecture distribuée.
- des moyens de traitement, sur lesquels repose la distribution du système d'automatisation.
- des moyens de mémorisation, qui autorisent la distribution de ses bases de données et qui comportent, en particulier, des informations de conduite, de maintenance, de gestion technique, etc.

Ainsi, un schéma général est donné figure 2.1. Cette représentation est proche de celle d'une machine informatique classique, elle va permettre de définir la notion de service.

Les interfaces sont les structures par l'intermédiaire desquelles l'instrument intelligent interagit avec son environnement. Elles regroupent les interfaces avec les opérateurs, les automatismes, le processus physique. Les informations traversant ces interfaces sont soit des signaux physiques, soit des données sous forme codée (trame sur un bus de terrain par exemple).

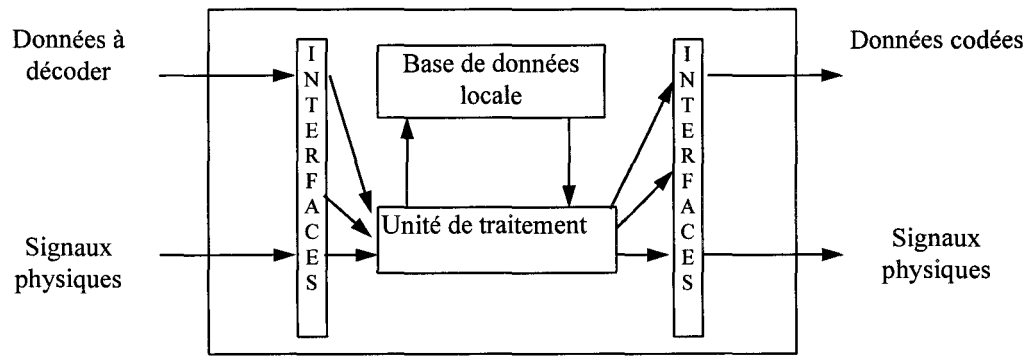


figure 2.1 : Architecture matérielle d'un instrument intelligent

2.3. Services d'un instrument intelligent

2.3.1. Définition

Nous partons du principe que la réalisation d'un service est demandée dans un but déterminé, ainsi :

Définition 2.1 :

> En se plaçant d'un point de vue externe à l'instrument intelligent, un service est le résultat de l'exécution d'un traitement (ou d'un ensemble de traitements), auquel on peut associer une interprétation en termes fonctionnels.

D'après cette définition, la description d'un service consiste à décrire le résultat produit par son exécution. Celui-ci se traduit par un effet perceptible au niveau de la base de données locale (exemple :mise à jour des variables mesurées pour un capteur) et/ou au niveau de ses interfaces de sortie (exemple :signal de commande d'une vanne ou autre pour un actionneur). Ainsi, du point de vue de l'utilisateur, la réalisation d'un service se traduit par la mise à jour des variables qu'il produit au moyen du traitement appliqué aux variables qu'il consomme.

2.3.2. Variables produites - variables consommées

Nous appelons, variables produites par un service les variables qu'il modifie dans la base de données locale et/ou qu'il présente sur une interface de sortie. Afin de préciser les valeurs obtenues il faudra décrire les traitements exécutés (traitement séquentiel, algorithmes etc.) et les variables sur lesquelles portent ces traitements, ces dernières sont dites : variables consommées. Par exemple, un service d'écriture consomme des données d'entrée prélevées sur l'une des interfaces et les mémorise dans la base de données locale qui se trouve ainsi modifiée. Un service de lecture prélève des données dans la base de données locale et les rend disponibles sur l'une des interfaces de sortie. D'autres services plus complexes, peuvent à la fois consommer et produire des données ou consommer et produire des signaux.

2.3.3. Durées de vie et dates de production d'une variable

Pour certains systèmes, des contraintes temporelles conduisent à associer à chacune des variables consommées un délai de péremption, définissant une date au delà de laquelle la

variable en question aura été générée par son producteur depuis un temps trop long, incompatible avec une utilisation pertinente par le consommateur. Pour les variables soumises à ce type de contraintes, le producteur leur associera également la dernière date de production.

2.3.4. Variables et paramètres

Il est parfaitement envisageable que le traitement réalisant un service soit paramétrable, ceci permettant de moduler les résultats de son exécution. Pour cette raison, il convient de distinguer parmi les variables consommées par un service celles que nous pourrons qualifier de paramètres. Cependant, la notion de paramètre reste très vague, et l'utilisation du vocable "paramètre" résulte davantage d'une appréciation intuitive que du respect d'une définition générale si tant est qu'elle puisse exister. En effet, on rencontre l'**usage** du terme "paramètre" pour désigner notamment :

- une variable inaccessible en mode de fonctionnement normal.

  Exemple : configuration, initialisation.

- une variable dont la modification présente par nature un caractère exceptionnel.

  Exemple : cadence de production.

- une variable fournie à l'occasion de la demande du traitement.

  Exemple : calcul de logarithme dans la base "p"

- une variable à relativement faible fréquence d'échantillonnage.

  Exemple : correction de l'influence de la température dans un capteur de pression.

La notion de paramètre semble donc suffisamment liée à son contexte pour qu'elle ne puisse être définie dans l'absolu. **Fondamentalement, il n'existe pas de différence entre paramètre et variable consommée**. En effet, le traitement élaborant la variable "s" en fonction de deux variables consommées "a" et "b" peut être représenté par l'expression s=f(a,b) sans qu'une distinction paramètre-variable n'y apparaisse.

Exemple : la tension aux bornes d'une diode à semi-conducteur est à la fois fonction du courant qui la traverse et de sa température. Le traitement effectué par ce composant peut correspondre à deux services différents :

- le premier, intitulé "mesure de température", est dit "paramétré" par le courant,

- le second, intitulé "fonction logarithme", l'est cette fois par la température.

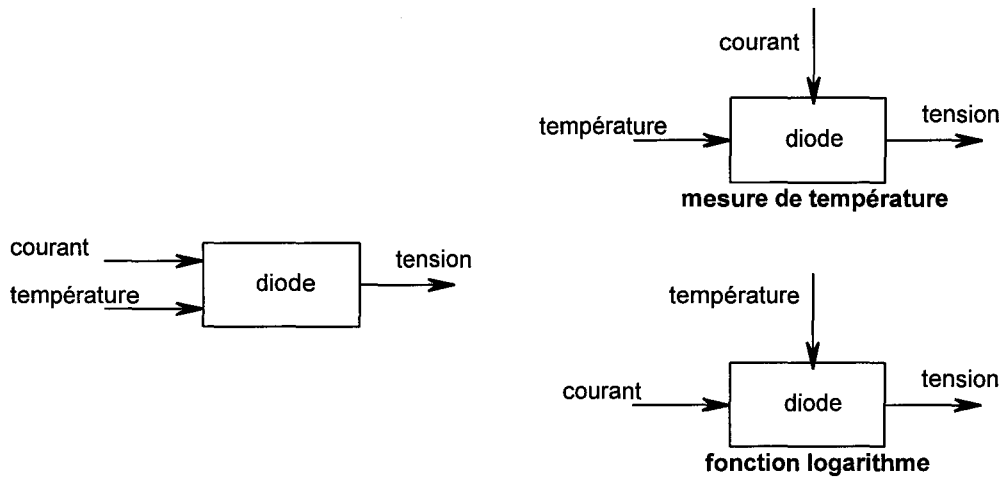


figure 2.2: Variables et paramètres

L'exemple précédent nous incite à considérer comme paramètres, des variables dites "plus lentes" que les autres.

Pour un traitement considéré, il est possible de représenter graphiquement les fréquences de mise à jour des variables à consommer sur une même échelle. Pour la majorité des traitements, cette représentation révèle l'existence d'un fossé entre deux catégories de variables, fossé qui pourrait alors servir de base à la notion de paramètre.

Exemple :

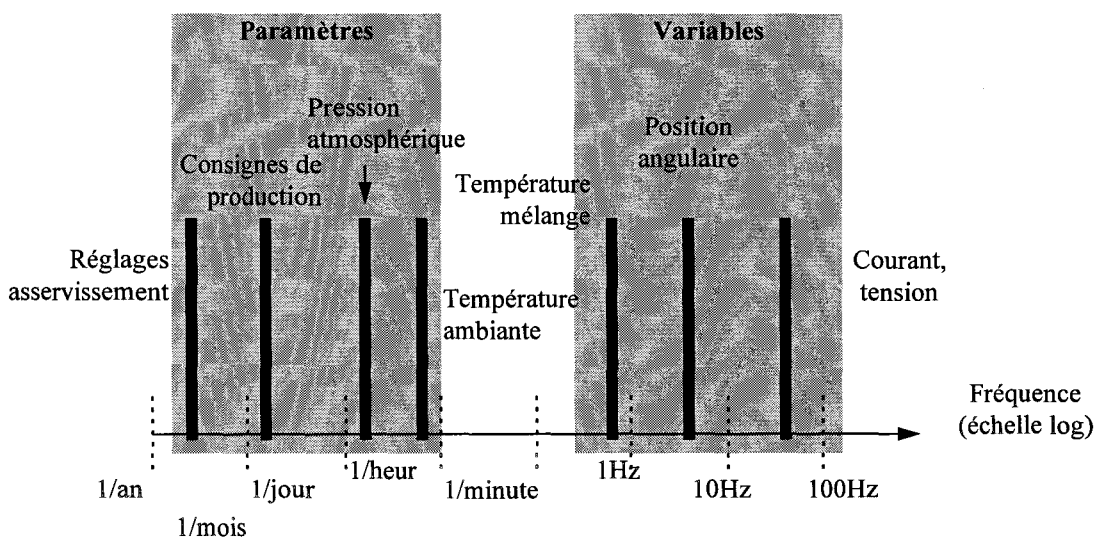


figure 2.2 bis: Variables et paramètres

Cette distinction se concrétise par le fait que contrairement aux variables "processus", les paramètres peuvent, dans ces conditions, se contenter de circuler de façon apériodique sur un réseau de communication tel que "WorldFip".

Dans notre spécification d'un service, nous ressentons le besoin d'effectuer cette distinction sans pour autant pouvoir la définir plus précisément que sous cet aspect fréquentiel.

Les réalisateurs d'un traitement donné restent *a priori* les plus compétents pour effectuer cette classification. Nous supposons donc que parmi les variables consommées par un service certaines peuvent être qualifiées de paramètres.

2.4. Propriétés d'un service

La réalisation d'un service se traduit par la mise à jour des variables produites au moyen d'un traitement appliqué aux variables consommées. Par définition, un service est caractérisé par les trois propriétés suivantes :

2.4.1. La réalisation d'un service est fonctionnellement atomique.

L'exécution d'un service est associée à une interprétation en termes fonctionnels. Une mise à jour partielle ou intermédiaire des variables produites n'a donc aucun sens : un service est une entité fonctionnelle indivisible.

2.4.2. La date de production de toutes les variables produites est unique.

Les mises à jour des différentes variables produites sont toutes supposées avoir lieu au même instant. Ce dernier correspond à l'instant de fin d'exécution du service.

2.4.3. La date de consommation de toutes les variables consommées est unique.

La date de consommation est supposée unique pour les différentes variables consommées. Cette date marque le début d'exécution du service.

Ainsi, un service peut être représenté par le schéma de la figure 2.3

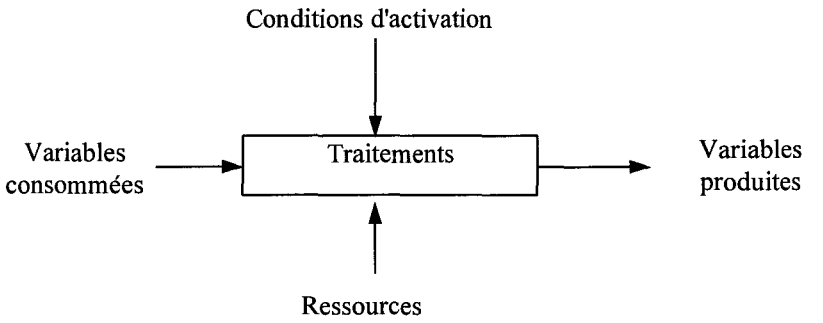


figure 2.3 : Représentation d'un service

L'exécution d'un service est déclenchée par la vérification d'une condition d'activation ; nous en détaillerons le mécanisme par la suite [§2.11].

Remarque :

Il est parfaitement envisageable qu'entre le début et la fin d'exécution d'un service la suspension du traitement réalisant le service ne soit pas interdite. Si c'est le cas, le service est dit préemptif, c'est-à-dire que son exécution peut être interrompue. En d'autres termes, si un service est préemptif, la date de mise à jour des variables produites peut-être différée.

2.5. Versions d'un service

2.5.1. Ressources, état des ressources.

La réalisation d'un service nécessite l'exécution d'un traitement appliqué aux variables qu'il consomme. Les variables consommées constituent les ressources informationnelles dont le service à besoin pour s'exécuter. De même, certaines ressources matérielles sont également nécessaires à l'exécution correcte d'un service : unité de traitement, pré-actionneur, etc.

Par ailleurs l'état de certaines ressources peut être connu grâce aux algorithmes de détection de défaillance qui localisent et diagnostiquent les défauts affectant ces ressources. Il en résulte donc un besoin de prévoir une gestion des différentes situations de panne que l'instrument est capable de détecter ; les paragraphes suivants détaillent ces mécanismes de gestion.

2.5.2. Service nominal ; service dégradé

Soit R(s) l'ensemble des ressources nécessaire à l'exécution d'un service s. Nous dirons que le service peut s'exécuter de manière nominale, à un instant donné, si l'ensemble des ressources, matérielles et informationnelles, qu'il utilise sont validées ou en état de fonctionnement normal à cet instant. Malheureusement, cela n'est pas toujours le cas et il est possible que certaines des ressources utilisées par le service soient défaillantes. Deux situations peuvent alors être distinguées :

- le concepteur n'a prévu aucun traitement de remplacement capable d'assurer le même service. Le service considéré devient alors indisponible.
- le concepteur a prévu au moins un traitement de remplacement n'utilisant pas les mêmes ressources que la version nominale. Dans le cas où ce traitement dispose effectivement des ressources qui lui sont nécessaires, il pourra être exécuté en lieu et place du traitement nominal, ceci lorsque les conditions de son activation seront réunies. La liste des services disponibles reste inchangée, mais le fonctionnement est dégradé.

En résumé, la défaillance de certaines ressources n'implique pas forcément l'indisponibilité du service qui les utilise. En effet des traitements de remplacement peuvent avoir été prévus. Pour un service donné l'ensemble de ces traitements définit les versions possibles de ce service. Chacune de ces versions est caractérisée par les ressources qu'elle nécessite.

2.6. Retour sur la définition d'un service

2.6.1. Définition

L'introduction de la notion de service dégradé permet de spécifier la robustesse de l'instrument intelligent face à la défaillance de certaines ressources. Cette notion de versions dégradées nous conduit à modifier légèrement la définition du service donnée au début de ce chapitre.

Définition 2.2 :

> Un service est un ensemble préordonné de versions, à chacune desquelles on peut associer la même interprétation en termes fonctionnels.

Soit V(s) l'ensemble des versions associées au service s. Chaque version est caractérisée par l'ensemble des ressources nécessaires à son exécution. Nous pouvons donc définir une fonction de l'ensemble des versions V(s) vers celui des sous-ensembles de ressource P[R(s)] telles que:

$V(s) \rightarrow P[R(s)]$

$v(s) \mapsto R[v(s)]$

Toutefois, les propriétés suivantes doivent être vérifiées :

1 Propriété de déterminisme

$\forall\ v_1, v_2 \in V(s)\ ;\ R(v1) \neq R(v2)$

2 Critère de classement

$\forall\ v_1, v_2 \in V(s) / R(v2) \subseteq R(v1)$ alors v_1 est préférée à v_2 et on note $v_1 > v_2$

La première propriété est évidente, elle exprime le fait que les sous-ensembles de ressources associées à deux versions différentes ne peuvent pas être identiques, autrement il y aurait un indéterminisme sur le choix de la version à mettre en exécution.

Les différentes versions d'un service sont classées selon un préordre de préférence. Chaque classe du préordre contient des versions qui représentent un état de fonctionnement plus dégradé que les versions de la classe qui précède. Ainsi, la version qui s'exécutera, à un instant donné, sera parmi celles possibles, la version de rang hiérarchique le plus élevé et dont les ressources nécessaires seront toutes présentes. Cette dernière version du service est dite version disponible du service, nous la notons v*(s). Par ailleurs, le classement des versions doit vérifier la deuxième propriété, autrement des versions du type v_1 ne seraient jamais mises en exécution.

Bien entendu, un service possède au moins une version, celle-ci définit deux classes : la classe de la version nominale et la classe vide où le service n'est pas disponible.

Il est difficile de définir un critère général permettant une classification des versions. L'exemple suivant illustre à quoi ce type de classification peut correspondre.

Exemple :

A titre d'exemple, considérons un service d'estimation de la vraie valeur d'une température à partir de trois organes sensoriels . La version nominale utilise les trois capteurs T1, T2, T3, qui en constituent des ressources nécessaires. A l'évidence, des versions dégradées n'utilisant que deux, voire un seul capteur, peuvent être mises en place. Le service d'estimation est alors décrit par la liste des versions suivantes :

Classe	Versions	Traitement	Ressources
0	nominale	T=(T1+T2+T3)/3	T1,T2,T32
1	V1	T=(T2+T3)/2	T2,T3
	V2	T=(T1+T3)/2	T1,T3
	V3	T=(T1+T2)/2	T1,T2
2	V12	T=T3	T3
	V13	T=T2	T2
	V23	T=T1	T1
classe vide	pas de version	∅	∅

La version qui s'exécutera, à un instant donné, sera parmi les sept versions possibles, celle dont les ressources nécessaires seront toutes présentes.

Nous formulons la classification des versions d'un service de la façon suivante :

Définition 2.3:

On note $D_i(s)$ la $i^{éme}$ classe des versions de s : $D_i(s) \subseteq V(s)$.

$D_i(s)$ définit le $i^{éme}$ degré de dégradation de s.

2.6.2. Relation avec les algorithmes de détection de défaillance

L'instrument intelligent est capable de surveiller l'état de certaines ressources qu'il utilise. Pour ce faire, il met en oeuvre des algorithmes de surveillance [CASS 92] lui permettant de détecter et de localiser les défauts affectant les ressources [STAR 96 a]. Ces algorithmes élaborent, pour chacune des ressources qu'ils surveillent, un indice de disponibilité qui évalue d'une façon binaire l'état de celle-ci. Les ressources dont la cohérence a été prouvée sont dites

validées. Ces informations permettent à l'instrument intelligent de gérer les versions des services et de répondre à une requête en exécutant la version de rang hiérarchique le plus élevé pour laquelle l'ensemble des ressources nécessaires sont disponibles. Si aucune version de la liste ne dispose des ressources qui lui sont nécessaires, le service considéré devient indisponible.

Définition 2.4:

On note dr l'indice de disponibilité de la ressource r.

$dr = 0$ ou 1

Pour une version donnée d'un service, les indices de disponibilité "dr" obtenus par les algorithmes de détection de défaillances et affectés aux différentes ressources, permettent de définir l'indice de disponibilité de la version considérée.

Définition 2.5:

On appelle indice de disponibilité d'une version v d'un service s,

$d(v) = \Pi_{r \in R(v)} dr$

Définition 2.6:

On dit qu'un service s est disponible et on note $d(s)=1$ ssi $\exists\ v \in V(s)$ tel que $d(v) = 1 \Leftrightarrow d(s) = d(v_1) \oplus d(v_2) \oplus \ldots \oplus d(v_n)$ tels que $v_i \in V(s)$

Par ailleurs, les différents indices de disponibilité permettent d'identifier, à tout moment, la version disponible d'un service ($v^*(s)$) c'est-à-dire celle qui s'exécutera en réponse à une requête. En effet v^* vérifie :

Propriétés

$v^*(s) = v(s)$ telle que $d(v(s)) = 1$ et $v(s) \in D_{min}$

avec D_{min} la classe (non vide) de degré de dégradation minimal.

Il est à noter que dans la réalité, les algorithmes de surveillance ne sont pas parfaits et peuvent déclarer disponibles des ressources qui ne le sont pas. Ainsi, le service que l'on croit disponible pourra être lancé en exécution et conduire à des résultats aberrants ou à des comportements dangereux. L'introduction des notions de sûreté de fonctionnement dans la conception des services et la minimisation de la probabilité de non-détection des procédures de surveillance, tendent à limiter ces effets. A l'opposé, les algorithmes de surveillance pourraient déclarer indisponibles des services dont les ressources sont, en réalité, en parfait

état de fonctionnement. Ce comportement, contraire à la disponibilité du système nécessite que l'on minimise autant que faire se peut la probabilité de fausse alarme.

2.6.3. Spécification d'un service

D'après ce qui précède, la description d'un service consiste à décrire le résultat obtenu en détaillant : les variables produites, les variables consommées, les versions possibles, pour chaque version les ressources nécessaires et les procédures appliquées. Afin de structurer la description de ces différentes notions, nous avons choisi de les représenter suivant la norme Backus Naur pour la spécification des protocoles de langages [BENZ 91]. Cette norme est celle utilisée pour spécifier le protocole du langage C++. Elle présente l'avantage de disposer de logiciels [PCCT] permettant de réaliser le compilateur d'un langage quelconque dont les règles sont décrites dans cette norme. Il est ainsi, possible à une machine informatique de reconnaître toute instance du langage en question.

Selon la norme Backus Naur les règles de spécification d'un service donné sont les suivantes :

<service>:=

<nom de service>

<liste des versions>:={<version> ; <classe de dégradation>}

<version>:=

<liste des paramètres >:={<variable>}

<liste des variables consommées>:={<variable > ; <délai de péremption>}

<liste des variables produites>:={<variable >}

<procédure>

<liste des ressources>:={<nom de ressource>}

<variable>:=

<nom de variable>

<type>:= <C++ type>

<classe de dégradation>:= "entier naturel"

2.7. Organisation des services : Modes d'utilisation

2.7.1. Définition des modes d'utilisation

L'ensemble des services prévus par le constructeur caractérise, d'un point de vue externe, le fonctionnement de l'instrument intelligent. En poursuivant le parallèle avec une machine informatique, cet ensemble peut être comparé au jeu d'instructions d'un microprocesseur. Cependant, l'analogie s'arrête là : alors que les instructions d'un microprocesseur doivent être organisées a priori dans un programme qui sera ensuite exécuté, les demandes de services peuvent être adressées à l'instrument à n'importe quel moment. On conçoit tout de même que cette liberté ne saurait être totale, et que, par exemple, un actionneur intelligent ne peut pas être en maintenance et en même temps proposer des services liés à la production. Dans ce sens, les conditions d'activation peuvent traduire l'autorisation d'accès et les conditions de sécurité nécessaires à l'exécution de tel ou tel service, selon la phase du cycle de vie et l'état de l'instrument intelligent.

Dans le cas d'un actionneur, par exemple, le service de commande de vérification d'ouverture, émanant d'un opérateur de maintenance, ne peut être exécuté qu'à condition que l'ensemble des services de configuration ait été réalisé au moins une fois, et que l'actionneur ne soit pas en mode automatique. Si on intègre toutes ces données au niveau des conditions d'activation, on s'aperçoit d'une part que celles-ci deviennent vite complexes, d'autre part que certains services ont des conditions d'activation similaires. Une solution consiste à organiser l'ensemble des services en sous-ensembles cohérents, qui correspondent à des modes d'utilisation. Ainsi, par exemple, tous les services ne pouvant être exécutés qu'après une phase d'initialisation, ne seront accessibles que lorsque cette étape sera terminée, c'est-à-dire qu'ils appartiendront à des modes d'utilisation inaccessibles avant le passage dans le mode "initialisation". Ceci simplifie l'expression des conditions d'activation, l'appartenance à un mode d'utilisation constitue en elle-même une condition vérifiée ou non.

Définition 2.7:

Un mode d'utilisation est un sous ensemble de l'ensemble des services de l'instrument intelligent. Un mode d'utilisation comprend au moins un service et chaque service appartient à au moins un mode d'utilisation.

Soit M l'ensemble des modes d'utilisation

M= {mj , j∈J} (J un ensemble d'indices)

Soit S l'ensemble des services

S = {si, i∈I} (I un ensemble d'indices)

Soit Sj ={si, i∈Ij} (Ij ⊂ I) l'ensemble des services proposés par le mode mj.

Un mode d'utilisation est caractérisé par

1 : ∀ j ∈ J, Sj ≠∅ .

2 : ∀ si ∈ S, ∃ j ∈ J / si ∈ Sj .

3 : ∪j∈ J Sj = S

Ainsi, l'ensemble des modes d'utilisation est un recouvrement de l'ensemble des services (figure 2.4).

A un instant donné, l'instrument se trouve dans un mode d'utilisation donné. Seuls les services appartenant à ce mode pourront être exécutés (s'ils sont disponibles), toutes les requêtes de services hors du mode courant étant automatiquement rejetées. Une telle organisation des services joue alors le rôle de filtre : l'instrument intelligent n'acceptera que les requêtes des services cohérents avec le mode d'utilisation dans lequel il se trouve, ce qui accroît sa sûreté de fonctionnement.

Cependant, se posent les deux questions suivantes :

1) comment définir les modes d'utilisation ?

2) comment changer de mode d'utilisation ?

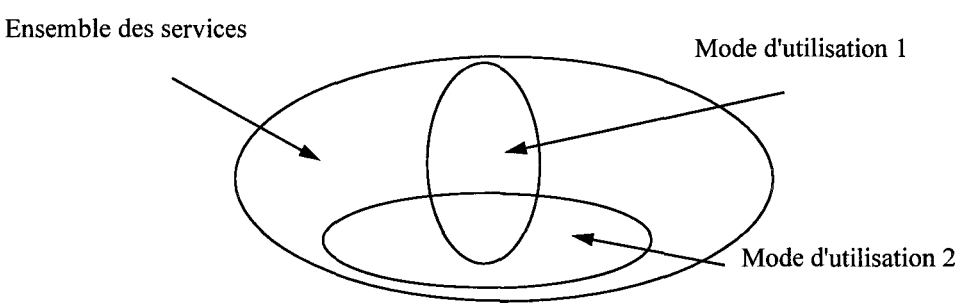


figure 2.4 : Mode d'utilisation et services

2.7.2. Structuration des services en modes d'utilisation

A priori, la structuration des modes d'utilisation est parfaitement arbitraire. En effet, aucune raison ne milite pour un découpage de l'ensemble des services plutôt qu'un autre. Cependant,

l'idée de cohérence des services qui constituent un mode d'utilisation trouve sa justification à travers une classification relativement générale des types de fonctionnement possibles : hors service, en configuration, en mode manuel, en mode automatique, etc. Chaque type de fonctionnement est relatif à une utilisation par un opérateur donné, dans une situation donnée (phase du cycle de vie). Un mode d'utilisation correspond ainsi à un ensemble de services mis à la disposition de cet opérateur dans cette situation. La notion de cohérence des services est donc interne à l'instrument et est relative au couple < situation ; opérateur >. Il s'agit d'une notion statique, qui peut d'ailleurs être différente d'un type d'instrument à l'autre. Une autre notion de cohérence - dynamique, celle-ci - peut être introduite vis-à-vis de l'installation [ch 6.]. En effet, l'interopérabilité des instruments intelligents suppose qu'un appareil A, émettant une requête vers un appareil B, le trouve dans un mode d'utilisation tel que cette requête ne soit pas rejetée. Ceci suppose une certaine standardisation des modes d'utilisation ou tout au moins un paramètrage de chaque appareil lui permettant de "connaître" l'organisation des services des appareils avec lesquels il devra interopérer. A un instant donné la cohérence dynamique est donc une cohérence relative aux modes d'utilisation des différents équipements en service. Elle est externe à l'instrument et concerne le système automatisé considéré dans son ensemble.

2.7.3. Changement de mode d'utilisation

A l'évidence, la liste des services de tout mode d'utilisation doit comporter un service spécifique de changement de mode, faute de quoi il deviendrait impossible de le quitter. Le mode d'origine étant bien sûr le mode courant, la requête de changement de mode doit indiquer le mode destination. Cependant, on ne peut pas rejoindre n'importe quel mode à partir de n'importe quel autre, pour des raisons de sécurité et de cohérence de fonctionnement. Ainsi, la requête de passage en mode automatique (positionnement avec régulation par exemple), devra être rejetée si les paramètres du régulateur n'ont pas été configurés. C'est pourquoi il convient de définir les conditions logiques qui permettent le passage d'un mode d'utilisation à un autre. La donnée de l'ensemble des modes et des conditions de passage définit entièrement la gestion des modes d'utilisation de l'instrument intelligent. Cette gestion peut être décrite par un graphe d'état (figure 2.5) ou, de manière équivalente, par un tableau des transitions portant en ligne les modes origine, en colonne les modes destination, et aux intersections les conditions logiques permettant le passage des uns aux autres (figure 2.6).

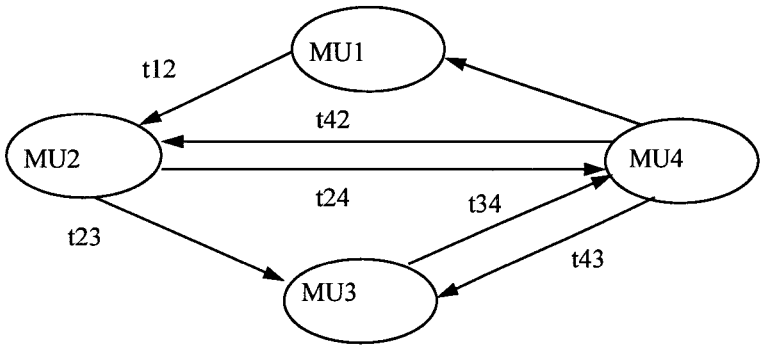


figure 2.5 : Graphe d'enchaînement des modes d'utilisation

	Conduite automatique	Conduite manuelle	Configuration	Repli
Conduite automatique	0	A	B	G
Conduite manuelle	D	0	C	0
Configuration	E	F	0	0
Repli	D	A	H	0

A : Requête émise par l'opérateur de conduite à partir du terminal local
B : Requête émise par l'opérateur de conduite à partir du poste de configuration
C : Requête émise par l'opérateur de conduite à partir du poste de configuration
D : Requête émise par l'opérateur de conduite à partir du poste de conduite ou par l'opérateur de maintenance à partir du terminal local
E : Requête émise par l'opérateur de configuration à partir du terminal local
F : Requête émise par l'opérateur de configuration à partir du terminal local
G Passage automatique lié à la détection d'un défaut (exemple, surpression dan un circuit)
H : Requête émise par l'opérateur de configuration à partir du terminal local

figure 2.6 : Modes d'utilisation, exemple

2.8. Modes de marche d'un instrument intelligent

2.8.1. Introduction à la notion de mode de marche : problématique

L'expression "modes de marche" est apparue lorsqu'on a voulu assurer un fonctionnement cohérent et optimal d'un Système Automatisé de Production (S.A.P.) dans toutes les situations pouvant se produire. Le but est l'identification de ces différentes situations afin de définir les comportements appropriés du système et surtout d'éviter les situations de blocage. En effet l'impact d'une omission durant la conception d'un système peut avoir des

conséquences néfastes durant son exploitation. Il n'est pas rare de constater, dans des systèmes hautement automatisés, que l'intervention humaine nécessaire est beaucoup plus importante que ce qui avait été initialement envisagé.

Part ailleurs, bien que la bibliographie sur les S.A.P. reconnaisse largement l'importance de l'analyse des modes de marche, il existe comme le signale [BILA 94], plusieurs interprétations différentes du concept de mode de marche. Pour cette raison, la majorité des études menées sur ce concept, proposent surtout des méthodes de prise en compte des modes de marche d'un SAP sans pour autant les définir [BILA 94] [BOIS 91]. Une tentative intéressante pour la détermination des modes de marches d'un SAP est le guide GEMMA proposé par [ADEPA]. Ce guide permet de s'interroger, lors de la conception, sur la nécessité de prévoir, ou non, l'un des modes de marche qu'il propose. En aucun cas, ce guide ne propose de méthodes formelles pour définir d'une manière précise et univoque cette notion de mode de marche.

En conclusion, nous sommes contraints de constater que les travaux menés sur le concept de mode de marche se caractérisent par:

- D'une part, l'inexistence d'une description formelle de la notion de mode de marche d'un équipement donné. Ce vide théorique est surtout du à ce que les travaux concernés ont toujours voulu faire abstraction totale du type d'équipement considéré. Rappelons que le problème des modes de marche est de déterminer les situations dans lesquelles peut se trouver un équipement donné. Pour cette raison, il est difficile de définir formellement la notion de mode de marche sans une description formelle du type d'équipement en question.
- D'autre part, ces travaux ne distinguent pas les deux types de situations possibles pour un équipement donné : celles dues à un fonctionnement normal de l'équipement et celles dues aux défaillances. En effet toute situation possible d'un équipement est le "produit" d'un état prévu par le fonctionnement normal et d'un état de défaillance subi.

Ainsi notre démarche pour aborder cette notion de mode de marche consistera à définir formellement les différentes situations dans lesquelles un instrument intelligent peut se trouver : nous appelons ces situations modes de marche. Pour identifier ces modes de marche,

nous couplons les différentes situations de fonctionnement normales avec celles relatives à des défaillances.

Toutefois, nous commençons cette partie par une brève étude bibliographique pour illustrer comment la notion de mode de marche est abordée en général.

2.8.2. Bibliographie sur le concept de mode de marche

2.8.2.1. Notion de mode de marche pour [BOIS 91]

La problématique des travaux de [BOIS 91] est l'intégration de la gestion des modes de marche dans le pilotage d'un système automatisé de production. Les travaux de S. Bois reposent sur l'hypothèse que le comportement individuel d'une machine, effective ou virtuelle, se décrit par un graphe unique et commun à toutes les machines. Ce graphe est structuré autour de six modes de marche inspirés du Gemma (figure 2.7).

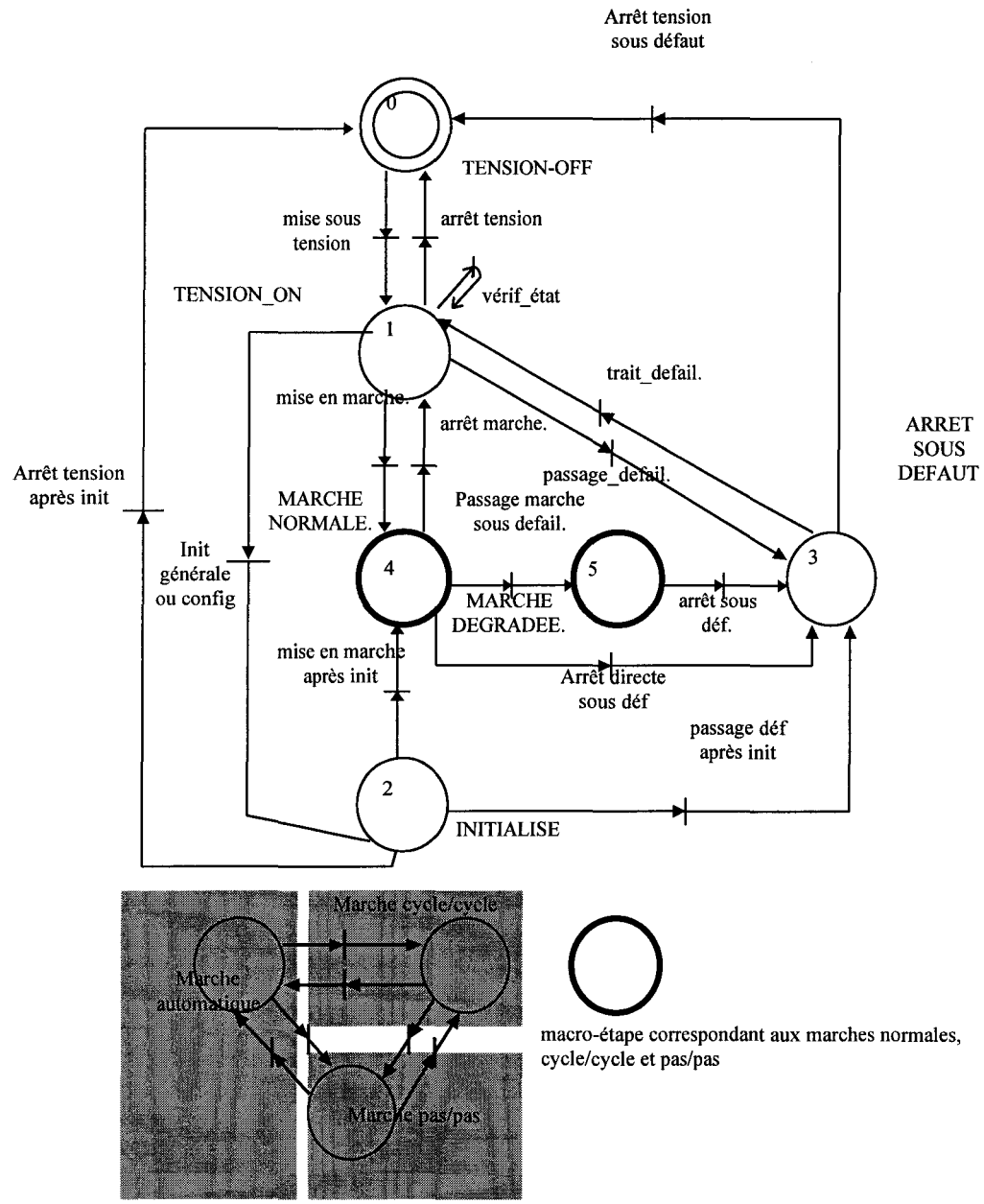


Figure 2.7: graphe d'état d'une machine

S. Bois considère que la gestion de ces états (pour toutes les machines d'un SAP) est un problème global qui doit prendre en compte les liens entre machines. Ainsi, ses travaux proposent une méthode de gestion des modes de marche tout en tenant compte des différentes contraintes entre états (nous reviendrons par la suite [ch 4] sur la notion de contrainte selon Bois).

Par ailleurs, il n'existe dans les travaux de [BOIS 91] aucune formulation de ce que peut être un mode de marche. Nous ne trouvons pas, non plus, de preuve que le graphe proposé représente bien l'ensemble des états de fonctionnement d'une machine quelconque.

2.8.2.2. Notion de mode de marche pour [BILA 94]

[BILA 94] pour sa part, ne cherche pas à définir les modes de marche d'un SAP. Ses travaux de recherche sur la modélisation des modes de marche d'un SAP se basent sur la bibliographie pour légitimer l'hypothèse d'existence d'un graphe d'état représentant les différents modes de marche d'un chaque SAP élémentaire. L'essentiel des travaux consiste à faire la composition des différents graphes pour obtenir et simplifier celui du système global composé de SAP élémentaires.

2.8.2.3. GEMMA

La célébrité du Gemma est telle que sa présentation n'est plus à faire, [BOIS 91] [BILA 94]. Rappelons simplement que le guide Gemma à été développé par l'ADEPA (Agence nationale pour le Développement de la Production Automatisée). Ce guide fournit une aide durant la conception d'un système automatisé de production et il est susceptible de permettre une étude complète des différents modes de marche envisageables. L'approche faite par le Gemma pour l'identification des modes de marche d'un SAP relève d'avantage d'une très grande expérience que d'une démarche méthodique. Pour cette raison même si le Gemma ne propose pas de méthode formelle il peut au moins être considéré comme un récapitulatif exprimant les besoins des utilisateurs potentiels. Ce dernier point fait la force du Gemma et légitime sa présence dans les différentes approches de la notion de mode de marche.

Part ailleurs, le Gemma reste très général et nécessite, pour chaque cas, d'approfondir les modes de marche prévus. En effet, pour chaque mode de marche, défini par le Gemma, un système peut être dans une infinité de situations possibles qui sont toutes à déterminer.

D'autre part, les définitions du Gemma peuvent conduire à des interprétations différentes selon l'utilisateur. En effet le fonctionnement normal pour l'opérateur de maintenance peut correspondre à un fonctionnement en mode test pour l'opérateur de conduite. Ainsi, nous constatons qu'il n'existe pas une interprétation unique qui éviterait toute confusion.

2.8.3. Modes de marche d'un instrument intelligent

2.8.3.1. Etat choisi et état subi

Nous avons vu que, pour qu'un service d'un instrument intelligent soit disponible, deux conditions doivent être remplies :

- D'une part, le service doit appartenir au mode d'utilisation en cours. Celui-ci résulte de l'exécution d'une requête de changement de mode, exécutée selon le mécanisme décrit par le graphe d'état des modes d'utilisation. Le mode d'utilisation en cours est le résultat d'une action volontaire des utilisateurs ou d'un fonctionnement prévu de l'instrument intelligent, c'est donc un état de fonctionnement choisi.
- D'autre part, il doit exister une version n'utilisant que les ressources non défaillantes à l'instant courant. L'ensemble des ressources défaillantes résulte d'événements sur lesquels les utilisateurs n'ont pas de prise et qu'ils subissent d'une manière involontaire.

D'après ce qui précède, il nous semble évident de distinguer deux types d'états pouvant définir les situations dans lesquelles peut se trouver un instrument intelligent. Nous avons d'une part le mode d'utilisation qui correspond à un état prévu par le fonctionnement normal et qui propose un ensemble de services donné. D'autre part, nous avons les différentes situations de dégradations dues aux défaillances des ressources, ces situations permettent de définir la version disponible (v* [§2.6.2]) de chaque service.

Ainsi, les différentes situations dans lesquelles un instrument intelligent peut se trouver à un moment donné sont le résultat des dégradations (défaillances) subies par les ressources et d'actions volontaires des utilisateurs qui conduisent à un mode d'utilisation particulier. Ces différentes situations d'un instrument intelligent nous informent sur les fonctionnalités disponibles ainsi que leur état de dégradation.

2.8.3.2. Mode de marche associé à un mode d'utilisation :

Les algorithmes de détection de défaillance qui peuvent être implantés [§2.6.2] permettent d'obtenir à un instant donné pour chaque service d'un mode d'utilisation, la version qui sera

exécutée. Pour le mode d'utilisation en question, l'ensemble de ces versions définit son état de dégradation.

Nous appelons donc mode de marche d'un instrument intelligent : la liste des versions disponibles des services d'un mode d'utilisation. La gestion des modes de marche permet à l'instrument intelligent de fournir cette information aux différents opérateurs chargés de la conduite de la maintenance, de la gestion du processus ...

Définition2.8:

On appelle mode de marche d'un instrument intelligent associé à un mode d'utilisation Mu :

$Mm(Mu) = \{v^*(s) / s \in Mu\}$

On a donc $Mm(Mu) \in \Pi_{s \in Mu} V(s)$

On appelle mode de marche nominal de Mu :

$Mu^\circ = \{v^\circ(s) / s \in Mu$ et $v^\circ(s)$ la version nominale de $s\}$

Par ailleurs, un mode d'utilisation a été défini comme un ensemble de services cohérents. Il est donc parfaitement envisageable de considérer que, en l'absence de certains de ses services un mode d'utilisation donné perd toute signification. Le mode "marche automatique" d'une vanne de régulation, par exemple, perd son sens en cas de défaillance de la régulation. D'une manière générale, nous pouvons envisager qu'à partir d'un certain degré de dégradation global de ses services, un mode d'utilisation donné perd toute signification. Par conséquent, il est parfaitement envisageable de considérer que parmi les modes de marche de l'instrument qui peuvent être associés à un mode d'utilisation considéré, certains ne soient pas admissibles.

Dès lors, on peut diviser les modes de marche, qui peuvent être associés à un mode d'utilisation donné, en deux ensembles : le premier définit le mode de marche nominale et ceux dégradés, le deuxième définit les situations d'indisponibilité du mode d'utilisation considéré. Cependant, cette distinction entre les différents modes de marche possibles est étroitement liée aux types de services, ce qui rend toute formulation générale très difficile. Nous pouvons simplement conclure qu'il existe une condition logique entre les différentes versions des services d'un mode d'utilisation, condition qui permet de déterminer si un mode de marche associé au mode d'utilisation est autorisé ou non.

2.8.3.3. Situations imperceptibles par l'instrument

Nous avons dit qu'un mode de marche se définit par un mode d'utilisation particulier et un état de dégradation des ressources. Cependant, les algorithmes de détection de défaillances, ne permettent, à l'instrument intelligent, de percevoir que la défaillance des ressources qu'ils surveillent. Par conséquent, il peut exister des modes de marche imperceptibles par l'équipement, dus aux dégradations des ressources non surveillées. Il se pose alors le problème d'identification et de la prise en compte des situations ou modes de marche en question. Ces différentes situations se situent aux delà des possibilités techniques ("de l'intelligence") offertes par l'équipement. Il est important dans ce cas d'étudier dès la conception l'impact de ces situations et de prévoir si nécessaire une intervention humaine pour les identifier lors du fonctionnement.

2.9. Spécification d'un mode d'utilisation

Les règles de spécification des modes d'utilisation d'un instrument intelligent ainsi que leur organisation, peuvent être décrites suivant la norme Backus Naur.

<graphe des modes d'utilisation>:=

<liste des modes d'utilisation>:={<mode d'utilisation>}

< liste des transitions >:={<transition>}

Les éléments terminaux ont les descriptions suivantes :

<mode d'utilisation>:=

<nom du mode>

{<service>}

{<moyens d'accès au mode>}

<liste des modes de marche autorisés>:={<mode de marche>}

<mode de marche>:= {<service du mode ; version disponible>}

<transition>:=

<mode de départ>:=<nom du mode>

<mode d'arrivée>:=<nom de mode>

<condition de tir>:="expression logique"

2.10. Requêtes - Langage et protocole de commandes

2.10.1. Requêtes des utilisateurs

L'exécution d'un service est le résultat d'une demande de service, appelée requête. Ainsi, le rôle de l'instrument intelligent est d'interpréter et d'exécuter les requêtes qui lui sont transmises par les différents utilisateurs. Une requête est définie par :

- son nom, qui permet de l'identifier et d'identifier le service correspondant,
- ses paramètres d'exécution, qui permettent d'avoir des résultats modulables,
- son origine et son mode de transmission, c'est-à-dire l'identification de l'entité qui l'a produite (opérateur de conduite ; de maintenance ; de gestion ; poste de supervision ; automate ; calculateur ; etc.) et de l'entité qui l'a communiquée (face avant, réseau, terminal local)

L'ensemble des requêtes paramétrées auxquelles l'instrument intelligent est capable de répondre définit son langage de commande. L'ensemble des origines et des modes de transmission autorisés pour les requêtes du langage de commande définissent un protocole de commande.

2.10.2. Spécification du protocole de commande

Par ailleurs, la spécification d'un service permet d'identifier l'ensemble des requêtes concernant l'exécution du service considéré [§2.6.3], ainsi pour définir le protocole de commande d'un instrument intelligent il suffit de spécifier pour chaque service les utilisateurs et les modes de transmission autorisés. Par conséquent la spécification du protocole de commande d'un instrument intelligent peut se faire de la façon suivante :

<Protocole de commande> := {<service>, <liste d'utilisateurs>}

<liste d'utilisateurs> :={<nom d'utilisateur>, <mode de transmission>}

2.11. Gestion des services

2.11.1. Etats d'un service

L'exécution d'un service repose sur un certain nombre de conditions. La vérification ou non de ces conditions va nous permettre de définir les différents états dans lesquels un service peut se trouver.

Pour qu'un service puisse être exécuté, il est tout d'abord indispensable qu'il appartienne au mode d'utilisation en cours. L'exécution d'un service est également fonction de l'état des ressources nécessaires aux traitements réalisant ce service. Un service est disponible si au moins une de ses versions peut être exécutée. Dans le cas contraire, le service est indisponible.

On a vu que l'exécution d'un service dépend, en plus de l'état de ses ressources matérielles, de la disponibilité des données qu'il consomme (ressources informationnelles). Ainsi, un service peut devenir indisponible, alors que ses ressources matérielles sont en bon état, par le fait qu'un des producteurs d'une information qu'il consomme est inopérationnel.

Un service disponible est actif dès qu'une requête relative à son exécution a été présentée et est valide (origine et mode de transmission autorisés). Il restera inactif dans le cas contraire.

Les trois conditions :

- le service appartient au mode d'utilisation courant
- il existe une version du service n'utilisant que des ressources disponibles
- le service est demandé par un utilisateur et via un moyen de transmission autorisés,

conduisent à la définition de l'ensemble des services actifs à un instant donné.

2.11.2. Exécution des services

Le service passe dans l'état actif, s'il est disponible, dès qu'une requête relative à son exécution a été présentée, et qu'elle est valide c'est-à-dire que son origine et son mode de

transmission sont autorisés. Cependant, il est possible qu'à un moment donné, plusieurs requêtes relatives à des services différents du même mode d'utilisation soient présentées. Pour cette raison, il convient de distinguer un service demandé mais pas encore exécuté (état en attente) d'un service en cours d'exécution (état en exécution). A un moment donné, plusieurs services peuvent être en attente, un seul sera en général en exécution. C'est le rôle d'un module de gestion spécifique (l'ordonnanceur) que de sélectionner, à la fin de l'exécution d'un service en cours, le service suivant à mettre en exécution parmi la liste des services qui sont en attente. Différentes stratégies peuvent pour cela être mises en oeuvre [LIST 83].

La figure 2.8 résume l'organisation des différents états dans lesquels peut se trouver un service.

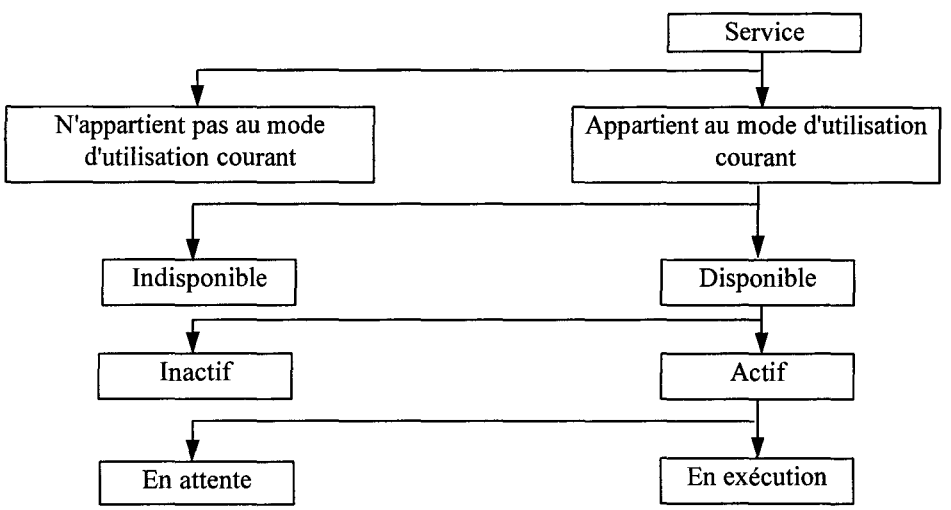


figure 2.8 : Organisation des états d'un services

2.11.3. Gestion des modes de marche et gestion des activités

On distingue deux niveaux de gestion des services : la gestion de modes de marche et la gestion des activités.

La gestion des modes de marche repose d'une part sur la gestion des modes d'utilisation et d'autre part sur la présence dans l'instrument intelligent d'algorithmes de surveillance permettant d'en déterminer l'état. Elle recouvre l'ensemble des mécanismes qui apparaissent figure 2.9.

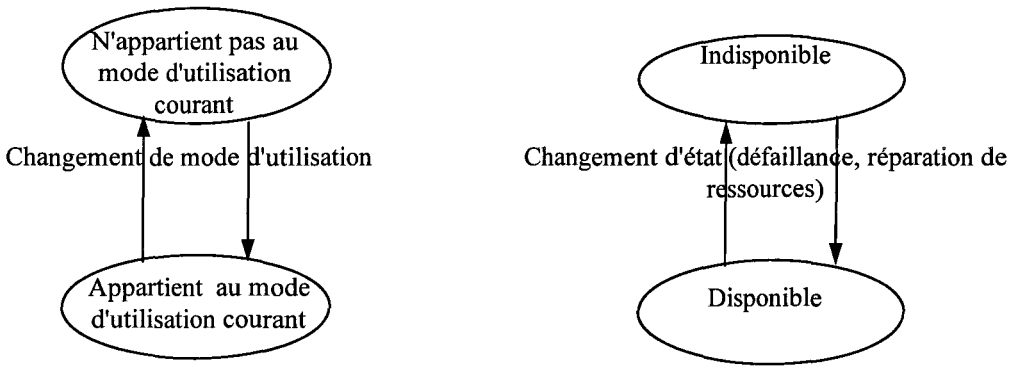


figure 2.9 : Mécanisme de transition entre états
(gestion des modes de marches)

La gestion des activités recouvre l'ensemble des mécanismes qui apparaissent figure 2.10 . Elle comprend deux niveaux. Le premier constitue la liste des services en attente d'exécution. Pour ce faire, il évalue pour chacun des services disponibles du mode d'utilisation en cours, sa condition d'activation. Celle-ci est le produit de trois termes :

Requête présente : le service a-t-il été demandé ?

Origine autorisée : l'entité ayant demandé le service était-elle autorisée à le faire ?

Mode de transmission autorisé : le mode de transmission de la requête correspond-il à un moyen de dialogue autorisé ?

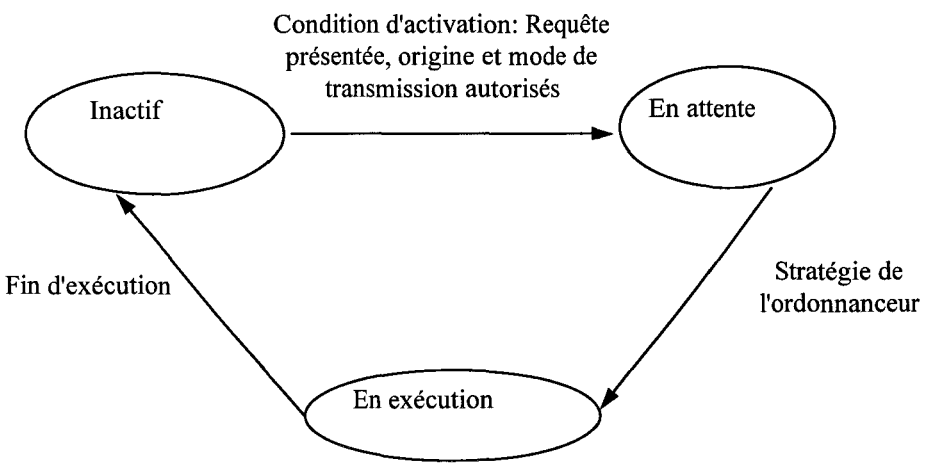


figure 2.10 : Mécanisme de transition entre état
(gestion des activités)

Le deuxième mécanisme de la gestion des activités recouvre la tâche d'ordonnancement. Cette tâche détermine, à un instant donné, quel est le service en exécution parmi la liste des services en attente. Lorsque l'exécution d'un service se déroule sans qu'elle puisse être suspendue on dit que le service est non préemptif. Dans le cas contraire, l'ordonnanceur gère non seulement la liste des services en attente mais aussi la liste des services suspendus. L'état suspendu doit alors être ajouté au graphe des états comme le montre la figure 2.11.

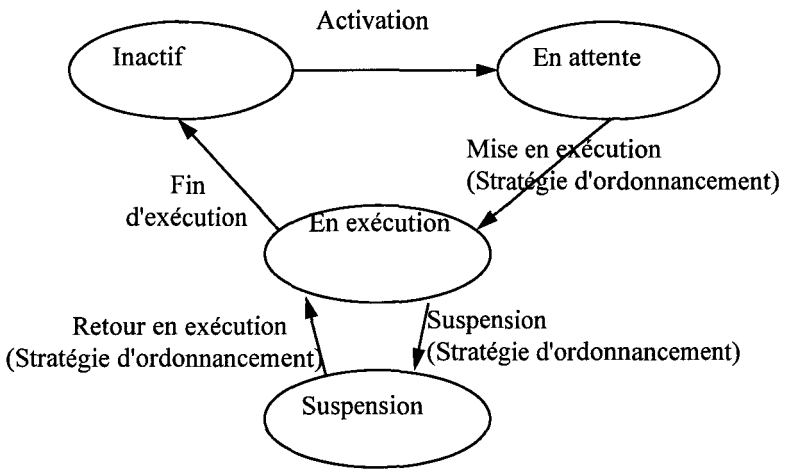


figure 2.11 : Mécanisme de transition entre états dans le cas d'un service préemptif

La notion d'exécution que nous venons de présenter est l'exécution immédiate, unique : comme cela apparaît figure 2.9 ou figure 2.10, un service disponible passera en attente dès qu'il sera demandé, en exécution lorsque l'ordonnanceur le décidera, et deviendra inactif lorsque l'exécution sera terminée. Il est possible de prendre en compte d'autres types de requêtes, plus complexes [STAR 96 a].

2.12. Modèle externe de spécification d'un instrument intelligent

Afin de définir des règles de spécification d'un instrument intelligent et constituer ainsi un support à une CAO du contrôle/commande d'applications distribuées interconnectant ces instruments au moyen de réseaux de communication, nous avons choisi de structurer les différentes notions du modèle externe de spécification d'un instrument intelligent, tout d'abord suivant la norme Backus Naur pour la spécification des langages et ensuite dans une représentation arborescente permettant une vision globale.

2.12.1. Représentation dans la norme Backus Naur pour la spécification des langages

Rappelons que Backus Naur est une norme pour la spécification des protocoles de langages [BENZ 91] Pour plus de détail sur cette norme, nous renvoyons le lecteur aux travaux de BENZAKEN [BENZ 91] où cette norme a été bien détaillée. Toutefois, rappelons aussi que cette norme est utilisée pour spécifier le protocole du langage C++ et qu'elle présente le grand avantage de disposer de logiciels [PCCT] permettant de réaliser le compilateur de tout langage dont les règles ont été décrites dans cette norme. Il est ainsi possible à une machine informatique de reconnaître toute instance du langage en question. Notons en plus que le logiciel [PCCT] est disponible sur le "word wide web".

Description du modèle externe :

<instrument>:=

<nom de l'instrument>

<graphe des modes d'utilisation>:=

<liste des modes d'utilisation>:={<mode d'utilisation>}

< liste des transitions >:={<transition>}

<Protocole de commande> := {<service>, <liste d'utilisateurs>}

Les éléments terminaux ont les descriptions suivantes :

<mode d'utilisation>:=

<nom du mode>

{<service>}

<liste des modes de marche autorisés>:={<mode de marche>}

<mode de marche>:= {<service du mode ; version disponible>}

<transition>:=

<mode de départ>:=<nom du mode>

<mode d'arrivée>:=<nom du mode>

<condition de tir>:="expression logique"

<service>:=

<nom de service>

<liste des versions>:={<version> ; <classe de dégradation>}

<version>:=

<liste des paramètres >:={<variable>}

<liste des variables consommées>:={<variable > ; <délai de péremption>}

<liste des variables produites>:={<variable >}

<procédure>

<liste des ressources>:={<nom de ressource>}

<variable>:=

<nom de variable>

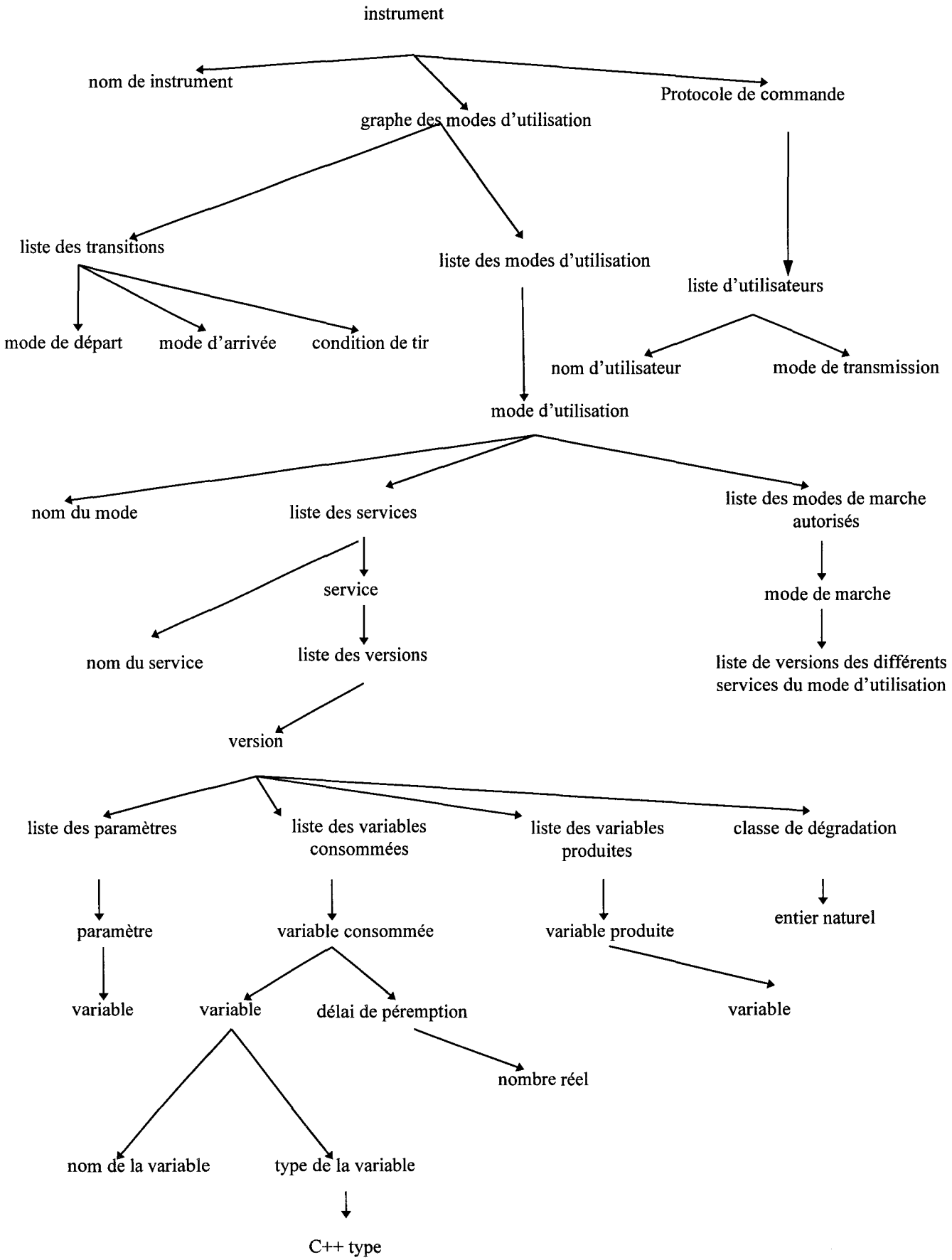
<type>:= <C++ type>

<classe de dégradation>:= "entier naturel"

<liste d'utilisateurs> :={<nom d'utilisateur><mode de transmission>}

2.12.2. Représentation arborescente

Les arbres et arborescences [GOND 79] sont des structures fondamentales de la théorie des graphes. Ces structures sont principalement utilisées pour l'analyse de données, parmi les applications les plus intéressantes on distingue : la classification automatique de données ; l'établissement de questionnaire d'identification ; etc. Dans ce mémoire, nous nous contenterons d'utiliser ce type de représentation uniquement dans le but de permettre une vision globale de l'organisation des différentes entités constituant le modèle externe de spécification d'un instrument intelligent. L'arbre obtenu est représenté par la figure suivante :


instrument
nom de instrument
graphe des modes d'utilisation
Protocole de commande
liste des transitions
liste des modes d'utilisation
liste d'utilisateurs
mode de départ
mode d'arrivée
condition de tir
nom d'utilisateur
mode de transmission
mode d'utilisation
nom du mode
liste des services
liste des modes de marche autorisés
service
mode de marche
nom du service
liste des versions
liste de versions des différents services du mode d'utilisation
version
liste des paramètres
liste des variables consommées
liste des variables produites
classe de dégradation
paramètre
variable consommée
variable produite
entier naturel
variable
variable
délai de péremption
variable
nombre réel
nom de la variable
type de la variable
C++ type

2.13. Conclusion

Nous avons défini dans ce chapitre un modèle générique à partir duquel il est possible de décliner la description particulière d'un instrument intelligent tel qu'il est vu par un utilisateur externe. Nous appelons ce modèle générique : modèle externe de spécification d'un instrument intelligent.

Le modèle proposé définit un instrument comme une entité proposant des services lesquels manipulent des variables et font appel à un ensemble de ressources. Toutefois, les services doivent être organisés en sous ensembles cohérents dits modes d'utilisation, cette organisation joue le rôle d'un premier filtre de commande pour éviter que l'utilisateur réalise des actions incompatibles.

Par ailleurs, les algorithmes de détection de défaillances qui peuvent être implantés nécessitent une gestion des différentes situations de panne qui peuvent se produire. L'identification de ces différentes situations et la définition des comportements appropriés nous ont conduits, d'une part à introduire la notion de versions d'un service, d'autre part à définir formellement la notion de mode de marche d'un instrument intelligent.

Dans la dernière partie de ce chapitre, nous avons récapitulé les différentes notions du modèle externe tout d'abord suivant la norme Backus Naur pour la spécification des langages et ensuite dans une représentation arborescente permettant une vision globale.

Deuxième partie
Intéropérabilité d'instruments intelligents au sein d'une application répartie

Chapitre 3. Organisation globale de la conduite d'un ensemble d'instruments intelligents

3.1. Introduction

Jusqu'à présent, nous avons considéré l'instrument intelligent comme une entité isolée. Nous avons vu comment spécifier la conduite et le comportement externe d'un instrument en organisant les services dans un graphe de modes d'utilisation. Cette démarche a abouti à la définition du modèle externe de spécification pour décrire le comportement individuel de chaque instrument intelligent.

Toutefois, la facilité d'interconnexion des instruments intelligents permet d'envisager l'analyse de la conduite collective d'un ensemble d'équipements dans le contexte où les services qu'ils proposent sont utilisés en vue de satisfaire des objectifs précis, c'est-à-dire de réaliser une application répartie particulière.

Dans ces conditions, nous devons disposer d'outils pour spécifier les comportements d'une application répartie interconnectant des instruments intelligents. Les recherches ayant pour objet l'analyse des modèles et méthodes de développement d'applications réparties [SIMO 95] ont mis en évidence la nécessité de modéliser une application par trois types d'architectures.

- On distingue tout d'abord l'architecture support pour décrire l'ensemble des composants fournissant des ressources matérielles et logicielles.
- Ensuite, on distingue l'architecture fonctionnelle, celle-ci décrit d'une façon abstraite le comportement externe. L'architecture fonctionnelle est donc un modèle abstrait exprimé de façon formelle de la structure et du comportement de l'application.

- On distingue enfin l'architecture opérationnelle qui désigne le résultat d'une projection ("mapping") de l'architecture fonctionnelle sur une architecture matérielle. En d'autres termes l'architecture opérationnelle désigne l'application elle-même une fois construite.

Dans ce chapitre, nous allons dans un premier temps introduire brièvement les trois types d'architectures citées plus haut, pour la modélisation d'une application. Ensuite, nous nous intéresserons au problème de la spécification des possibilités de conduite autorisées par une architecture support construite en interconnectant un ensemble donné d'instruments intelligents. L'objectif est de permettre, dans le prochain chapitre, de poser le problème de la réalisation d'une application particulière par un ensemble donné d'instruments intelligents.

3.2. Modèles d'architecture d'une application répartie

3.2.1. Architecture support d'une application

La littérature sur les applications réparties définit l'architecture support d'une application par l'ensemble de tous les composants qui fournissent les ressources matérielles et logicielles permettant la réalisation de l'application. Ainsi, une architecture support est caractérisée par une structure définissant les constituants élémentaires et leur interconnexion. Dans ces conditions, la spécification externe d'une architecture support particulière est caractérisée par :

- La liste des instruments interconnectés. Ces derniers représentent les constituants élémentaires souvent définis dans la littérature comme étant des équipements indépendants et autonomes reliés au réseau de communication [VEGA 96].
- La spécification externe de chaque instrument. Celle-ci permet de caractériser le comportement de chaque équipement. Le modèle proposé dans le chapitre précédent devra donc être instancié pour chaque équipement et spécifier ainsi les conduites possibles de chacun.
- La spécification de l'interconnexion (le système de communication est aussi un instrument [TRIC 97]).

3.2.2. Architecture fonctionnelle : finalité de l'application

A l'évidence, l'implantation d'une application sur une architecture support a pour objectif la réalisation d'une finalité précise.

Ainsi, la mise en oeuvre des services offerts par une architecture support sera généralement régie par des règles qui traduiront le comportement de l'application que l'on souhaite réaliser (exemple : procédure de démarrage/arrêt ; séquences obligatoires d'exécution de service).

Ces règles servent de base à la construction de la couche de mise en œuvre coordonnée des différents services offerts par les instruments qui composent l'architecture support.

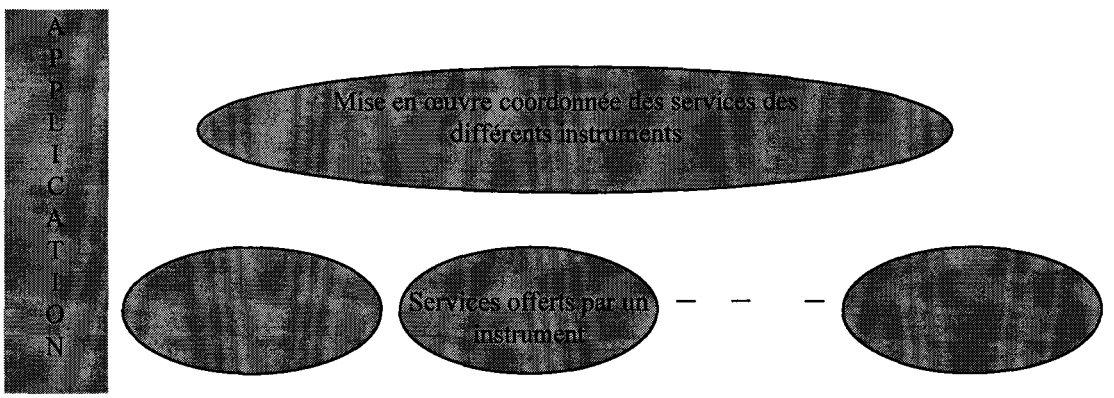


Figure 3.1 : réalisation d'une application

Par conséquent, la question qui se pose est de savoir comment se traduit la finalité d'une application répartie sur la conduite de son architecture support ? Dans la littérature, la finalité d'une application répartie est décrite par son architecture fonctionnelle. Cette dernière [VEGA 96] est définie par :

3.2.2.1. Les services utilisés par l'application.

Ils représentent l'ensemble des traitements ou services que l'application doit fournir ou accomplir. Ces services définissent un sous ensemble de l'ensemble des services proposés par l'architecture support.

3.2.2.2. Comportement spécifique (respect d'une architecture exécutive).

Ce comportement décrit comment l'application doit fournir et/ou accomplir les services proposés. Ainsi, la finalité d'une application peut imposer une architecture exécutive particulière

en imposant par exemple un ordre dans les exécutions de certains services. C'est le rôle de la couche de mise en œuvre coordonnée.

3.2.3. Architecture opérationnelle

Dans une démarche descendante, celle-ci est obtenue en projetant l'architecture fonctionnelle sur une architecture support particulière, qu'il convient de choisir. Cette démarche, et les problèmes qu'elle pose sont décrits dans [SIMO 95].

Nous nous intéressons dans le cadre de ce travail à la démarche ascendante : une architecture support étant donnée, est-il possible d'organiser la mise en œuvre coordonnée des services offerts par les composants élémentaires pour réaliser une architecture fonctionnelle particulière. Dans l'affirmatives, on dira que les composants élémentaires sont interopérables dans le cadre de l'application considérée. Dans la pratique, il est toujours possible de réaliser une architecture fonctionnelle quelconque mettant en œuvre les services offerts par une architecture support donnée : la plus ou moins grande facilité de prise en compte des contraintes imposées par l'architecture fonctionnelle conduira à des coûts de développement plus ou moins élevés de la couche de mise en oeuvre coordonnée des services. On pourra alors "mesurer" l'interopérabilité d'un ensemble de constituants élémentaires dans le cadre d'une application donnée, comme une fonction décroissante de la complexité de la couche de mise en œuvre coordonnée des services qu'ils offrent nécessaire au respect des contraintes imposées par l'architecture fonctionnelle de l'application considérée.

Nous pouvons distinguer deux cas de figures concernant la réalisation d'une application particulière à l'aide d'une architecture support donnée :

3.2.3.1. Conduite par simple pilotage des modes d'utilisation

C'est le cas le plus simple. Il suppose que l'application n'interdit pas la réalisation des services tels qu'ils sont proposés par l'architecture support, c'est-à-dire tels qu'ils sont organisés par les différents modes d'utilisation. Dans ces conditions, la réalisation de l'application peut se faire par simple pilotage des modes d'utilisation et par appel des services nécessaires. Ainsi, l'exploitation de l'architecture support dans le cadre de l'application ne nécessite aucun apport logiciel ou matériel autre que la couche de pilotage des modes d'utilisation, il est donc possible de conclure, dans ces conditions, à l'interopérabilité des instruments interconnectés.

3.2.3.2. Conduite par pilotage des modes d'utilisation et par élaboration d'une architecture exécutive propre à l'application.

Le deuxième cas, nécessite en plus du pilotage des modes d'utilisation, l'élaboration d'outils pour tenir compte des contraintes imposées par l'architecture fonctionnelle de l'application. Ainsi, la notion de mode d'utilisation n'est plus un filtre suffisant pour la conduite de l'application. Il existe donc un besoin d'affiner la notion de mode d'utilisation afin de spécifier l'architecture fonctionnelle de l'application donnée, qui ne résulte plus d'un simple "produit" entre les architectures fonctionnelles des constituants élémentaires de l'architecture support.

3.3. Composition d'instruments intelligents ou Spécification d'une architecture support.

3.3.1. Rappel sur la théorie des graphes Définitions des systèmes de transitions [BILA 94] [ARNO 92]

Nous introduisons ici les différentes notions de la théorie des graphes nécessaires à notre approche pour la spécification de l'organisation globale des services d'une architecture support construite autour d'un ensemble d'équipements.

3.3.1.1. Définition d'un graphe états transitions

Définition 3.1 : Graphe orienté [BILA 94] , [ARNO 92]

Un graphe orienté est un quadruplet (S ; T ; α ; β) où

- S est un ensemble de sommets, non nécessairement fini.
- T est un ensemble d'arêtes.
- α et β sont deux applications de T dans S qui, à chaque arête t, associent son sommet d'origine $\alpha(t)$ et son sommet but $\beta(t)$.

Définition 3.2 : Système de transition étiqueté [BILA 94] , [ARNO 92]

Un système de transition étiqueté sur un alphabet A est un quintuplet (S; T; α; β; λ) où

- (S ; T ; α ; β) est un graphe orienté : les éléments de S sont appelés états au lieu de sommets, les éléments de T sont appelés transitions au lieu d'arêtes.
- λ est une application de T dans A : $\lambda(t)$ est appelée l'étiquette de la transition t.

L'application (α; λ; β) de T dans $\underline{S}$ x A x $\underline{S}$ doit être injective : il ne peut y avoir deux transitions différentes de même origine, de même but et de même étiquette.

Ceci revient à considérer que T est une partie de $\underline{S}$ x A x $\underline{S}$ et permet de noter simplement ($\underline{S}$; T) un système de transitions (ou états transitions). Ainsi :

Un système états transitions étiqueté sur un alphabet A est un couple ($\underline{S}$; T) tel que

- $\underline{S}$ est un ensemble de sommets, non nécessairement fini.
- T est un ensemble de transitions T, c'est-à-dire est une partie de $\underline{S}$ x A x $\underline{S}$; $T \subseteq \underline{S} \times A \times \underline{S}$.

3.3.1.2. Composition de graphes : produit asynchrone

L'objectif est de modéliser par un seul graphe les différents états et transitions de plusieurs graphes en même temps. L'idée est de composer les ensembles des sommets de différents graphes de façon à définir un nouvel ensemble de sommets formé des n-uples ainsi définis. Les transitions seront obtenues par applications d'opérateurs appropriés. L'opération ainsi décrite est dite produit asynchrone de graphes états transitions. Cette opération a été utilisée par [BILA 94] pour modéliser le comportement d'un SAP composé à partir des automates modélisant le comportement de chacun des SAP élémentaires.

3.3.1.2.1. Définition des opérateurs / et \

Etant donné deux graphes états transitions G1($\underline{S}$1,T1) et G2($\underline{S}$2,T2) on définit les opérateurs suivants :

$$T1 / \underline{S}2 = \{ ((q,y), \sigma, (q',y)) , \forall (q, \sigma, q') \in T1 \text{ et } \forall y \in \underline{S}2 \}$$

$$\underline{S}2 \setminus T1 = \{ ((y,q), \sigma, (y,q')) , \forall (q, \sigma, q') \in T1 \text{ et } \forall y \in \underline{S}2 \}$$

Si (y,q) et (q,y) ont la même interprétation c'est-à-dire que quels que soient q et y ; (y,q) = (q,y) on a : $T1 / \underline{S}2 = \underline{S}2 \setminus T1$. C'est dans ce cas que nous nous placerons dans toute la suite de ce mémoire

3.3.1.2.2. Définition du produit asynchrone simple

Etant donné deux graphes états transitions G1($\underline{S}$1,T1) et G2($\underline{S}$2,T2), le produit asynchrone de ces deux graphes est défini par :

Définition 3.3

Soit G1($\underline{S}$1,T1) et G2($\underline{S}$2,T2) deux graphes états transitions

$G1 \times G2 = (\underline{S}, T)$ tel que

$\underline{S} = \underline{S1} \times \underline{S2}$

$T = T1 / \underline{S2} \cup T2 / \underline{S1}$

Exemple :

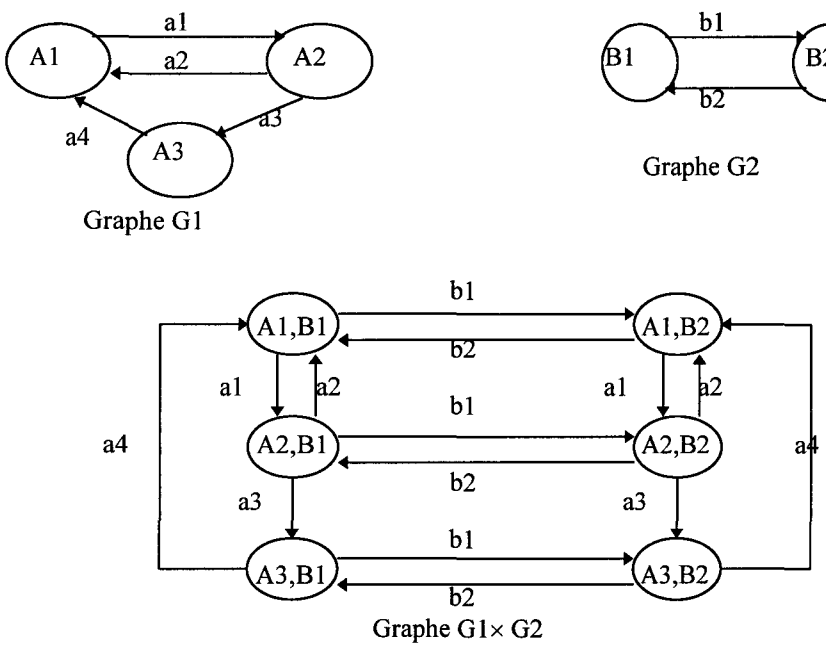


Figure 3.2: composition de graphes

Propriétés

Il est facile de voir que le produit asynchrone est commutatif.

Remarque :

Nous utiliserons l'opérateur Π pour désigner plusieurs produits asynchrones consécutifs.

3.3.2. Produit asynchrone de deux graphes de modes d'utilisation

Soit un couple d'instruments I1 et I2 ayant respectivement pour graphe des modes d'utilisation G1(M1,T1) et G2(M2,T2). Par définition même, le produit asynchrone G=G1×G2 est un graphe dont les sommets représentent des couples de sous-ensembles de services appartenant respectivement à M1 et M2. Chacun de ces sommets représente donc un ensemble de services proposés par le couple d'instruments (I1,I2). Par conséquence, G = G1 × G2 peut définir un nouveau graphe de modes d'utilisation qui représentent l'organisation globale des différents services de (I1,I2). Le fonctionnement de ce nouveau graphe est identique à celui décrit dans [§2.11.3] et relatif à un équipement unique. Les différents mécanismes de gestion du nouveau

graphe des modes d'utilisation sont faits automatiquement par I1 et I2 sans aucun autre apport matériel ou logiciel.

En conclusion, dans toute la suite, nous dirons, par abus de langage, que le produit asynchrone de deux graphes de modes d'utilisation d'instruments intelligents est lui-même un graphe de modes d'utilisation relatif au couple d'instruments considérés.

3.3.3. Protocole de commande d'un couple d'instruments intelligents.

Rappelons que le protocole de commande d'un instrument intelligent détermine quelles sont les origines autorisées (utilisateurs et moyens d'accès) pour émettre la requête d'exécution d'un service quelconque. Nous avons donc spécifié dans [§2.10.2] le protocole de commande d'un instrument intelligent par un ensemble énumérant pour chaque service l'ensemble des origines autorisées pour émettre une requête d'exécution. Si on note respectivement P(I1) et P(I2) les protocoles de commande des instruments I1 et I2, il nous est ainsi possible de définir le protocole de commande du couple d'instruments I1×I2 en réunissant les protocole de commande des deux instruments, on a donc : $P(I1 \times I2)=P(I1) \cup P(I2)$.

3.3.4. Composition d'instruments intelligents

Nous cherchons à spécifier dans un modèle unique la vision externe que l'on peut avoir d'un couple d'instruments intelligents, pour ceci il suffit de définir pour le nouvel instrument :

1) la liste de ses services,

2) le nouveau graphe des modes d'utilisation,

3) le protocole de commande,

Ainsi, étant donné un couple d'instruments I1 et I2 ayant respectivement pour graphe des modes d'utilisation G1 et G2 et pour protocole de commande P1 et P2, nous définissons l'instrument composé I= I1×I2 par :

Définition 3.4:

I= I1×I2 est tel que

1)l'ensemble des services de I est $S(I)=S(I1) \cup S(I2)$

2)le graphe des modes d'utilisation de I est $G = G1 \times G2$

3)le protocole de commande est $P(I1 \times I2)=P(I1) \cup P(I2)$

Remarque :

Comme dans le cas d'un instrument unique, plusieurs services peuvent être demandés simultanément.

Dans l'ensemble des instruments intelligent, la composition est une opération interne qui présente les propriétés suivantes:

Proposition 3.1 :

Commutativité : I1×I2= I2×I1

Associativité : I1×(I2×I3) = (I1×I2)×I3

Elément neutre : e tel que S(e) =∅ ; P(e)= ∅ ; et G(e) = (1 sommet, o arcs)

Démonstration :

Il est facile de voir que "×" est commutative, associative et admet "e" comme élément neutre

3.4. Conduite globale autorisée par une architecture support donnée

Nous avons défini dans le paragraphe précédent une loi de composition interne entre instruments intelligents. Cette loi de composition permet de déterminer la conduite globale autorisée par un couple d'instruments intelligents, ceci en spécifiant le modèle externe de l'instrument "produit". L'application de cette démarche à l'ensemble des instruments d'une architecture support permet donc de spécifier le modèle externe de cette dernière et ainsi de déterminer, grâce au nouveau graphe des modes d'utilisation, la conduite globale qu'elle autorise. Dans ces conditions, une architecture support peut être définie de la façon suivante :

Définition 3.5:

Soit $\underline{I}$ ={I1,I2, ...In} un ensemble d'instruments intelligents interconnectés

Chaque instrument est caractérisé par :

l'ensemble des services :S(I)

le protocole de commande :P(I)

et le graphe des modes d'utilisation G(I)

L'architecture support $\underline{I}$ construite en interconnectant {I1,I2, ...In} est définie par :

$S(\underline{I}) = \cup_i S(Ii)$

$P(\underline{I}) = \cup_i P(Ii)$

$G(I) = \Pi i\ G(Ii)$

Toutefois, notons que le fonctionnement global de ce nouveau graphe reste identique à celui décrit dans [§2.7] et relatif à un équipement unique. De même que dans [§2.11.3] le pilotage est fait automatiquement par les différents instruments, et ne nécessite aucun apport logiciel ou matériel. Ainsi, les exécutions des services sont obtenues sur requêtes spécifiques qui sont gérées par les équipements concernés.

3.5. Conclusion

L'intéropérabilité est le critère qui garantit qu'une coopération est possible entre deux instruments en provenance de constructeurs différents. La notion d'intéropérabilité est essentielle, car les utilisateurs comme les concepteurs souhaitent ne pas avoir à changer entièrement les supports logiciels de la conduite d'un ensemble d'équipements à chaque fois qu'ils ajoutent, suppriment ou remplacent un équipement.

Toutefois, si l'on part du principe que la conduite d'un ensemble d'instruments intelligents s'inscrit toujours dans le contexte où les services proposés sont utilisés pour satisfaire une finalité précise, on s'aperçoit que la possibilité d'une coopération entre les éléments d'un ensemble d'instruments peut dépendre de la finalité que l'on souhaite réaliser. En effet, la réalisation d'une application répartie par un ensemble d'instruments peut imposer des règles de conduite qui doivent être implantées dans une couche de gestion des équipements interconnectés.

En conclusion, la facilité d'interconnexion des instruments intelligents permet de construire des architectures support très diverses et de spécifier pour chacune d'elles la conduite globale qu'elle autorise au moyen du modèle externe d'un instrument "produit". Cependant, l'interopérabilité d'un ensemble d'instruments dans le cadre d'une application donnée dépend de la plus ou moins grande facilité avec laquelle les contraintes spécifiques de l'application peuvent être implantées. Il convient donc, et c'est l'objet du chapitre suivant de spécifier ces contraintes en relation avec le modèle externe d'instruments intelligents développé dans le chapitre 2.

Chapitre 4. Conduite d'un ensemble d'instruments intelligents pour la réalisation d'une application particulière : expression des contraintes entre requêtes

4.1. Introduction.

La problématique générale de ce chapitre concerne la conduite à adopter pour réaliser une application particulière par un ensemble d'instruments intelligents interconnectés. Plus précisément, il s'agit d'exprimer la répercussion au niveau de l'utilisateur des restrictions de conduite que peuvent imposer les contraintes propres à une application. Pour résoudre ce problème notre démarche est la suivante :

- Tout d'abord, nous postulons qu'une contrainte sur un service se traduit par l'existence d'une condition à respecter avant de proposer, ou non, le service aux utilisateurs. Ce postulat nous permet d'établir un lien entre la notion de contrainte et celle d'événement. Nous introduisons ainsi la notion de service activable à un instant donné.
- Ensuite, nous constatons que durant un intervalle de temps encadré par les instants d'occurrence de deux événements quelconques, l'ensemble des services activables par l'utilisateur ne varie pas. Nous appellerons ce type d'intervalle de temps "étape de fonctionnement" et l'ensemble des services activables "menu d'utilisation".
- Enfin, pour exprimer la chronologie selon laquelle les menus sont proposés à l'utilisateur, nous constatons que les menus et les événements peuvent être organisés dans un graphe états-transitions que nous appellerons graphe des menus. Ce graphe permet d'exprimer la répercussion des contraintes de l'architecture fonctionnelle sur la conduite globale d'une application.

Nous proposons aussi dans ce chapitre une méthode de construction du graphe des menus. Cette méthode aboutit à la description de ce graphe, ceci, en retrouvant à toutes les étapes de fonctionnement l'ensemble des services activables par l'utilisateur. Par ailleurs, la construction de ce graphe montre qu'il est possible que des menus d'étapes différentes soient constitués du même ensemble de services activables. Pour un utilisateur extérieur, il n'y a pas de différence entre deux menus proposant le même ensemble de services. Pour cette raison, nous fusionnons ce type de menus en définissant des classes d'équivalence et en levant les éventuelles indéterminations pouvant se poser. Une fois construit, le graphe des menus permet non seulement d'exprimer les contraintes mais aussi de retrouver les séquences de commande de l'application qui peuvent être automatisées. Ainsi, il nous sera possible dans le chapitre suivant de définir des services d'un niveau supérieur automatisant certaines séquences et permettant une commande plus élaborée de l'application considérée.

Nous commençons ce chapitre par une brève étude bibliographique sur la notion de contrainte afin de définir le contexte particulier dans lequel nous nous plaçons pour l'utilisation de ce mot.

4.2. Contraintes relatives à une application répartie : bibliographie et contexte d'utilisation de la notion de contraintes.

4.2.1. Introduction

La bibliographie sur la conception des systèmes automatisés souligne l'importance de la prise en compte des contraintes tout au long du cycle de vie d'un système [SHAR 95]. Toutefois, le mot "contrainte" est à utiliser avec précautions. En effet bien que dans l'absolu ce mot suppose une restriction ou une limitation quelconque, il n'a un sens que s'il est utilisé dans un contexte préalablement défini. Ainsi, une contrainte est une restriction imposée à l'un des attributs (caractéristiques) d'une spécification d'un système ou d'une structure quelconque [VEGA 96]. Pour cette raison, plusieurs travaux de recherche associent souvent au mot contrainte un adjectif supposé définir le contexte et l'attribut sur lequel porte la contrainte. On parle alors de :

- Contraintes temporelles [VEGA 96],

- Contraintes de fonctionnement [ELKA 93],
- Contraintes de production [BOIS 91],
- Contrainte d'ordonnancement [LIST 83]

Dans ce paragraphe nous allons définir le contexte et l'interprétation que nous donnons à la notion de contrainte. Par ailleurs et dans le but de permettre une comparaison, nous allons commencer par présenter quelques travaux de prise en compte des contraintes d'une application.

4.2.2. Notion de contrainte pour [BOIS 91]

La définition donnée par [BOIS 91] à la notion de contrainte est la suivante :

"Les contraintes sont les liens qui existent entre des machines lors de la production, introduites par les impératifs de l'automatisation avancée de la production. Elles permettent de définir et de spécifier les multiples dépendances qui existent entre les machines et leurs tâches".

En fait [BOIS 91] considère en s'inspirant du GEMMA [ADEPA], que chaque machine possède cinq états ou modes de marches structurés dans un graphe unique et commun à toutes les machines [§ 2.8.2]. Ainsi, la notion de contrainte porte ici sur les liens qui peuvent exister entre les états des différentes machines d'un système. [BOIS 91] considère qu'entre deux machines il existe deux types de contraintes.

- le premier est celui des contraintes entre états. [Bois] considère que leur nombre est de quatre :
    1 Contrainte de coopération (CC), elles expriment la nécessité pour chaque machine de la présence d'une autre machine,
    2 Contrainte de coopération partagée (CCP) pour exprimer le partage de ressources,
    3 Contrainte d'exclusion (CE), afin d'exprimer les exclusions mutuelles,
    4 Contrainte structurelle (CS) pour exprimer le fonctionnement en série ou en parallèle de deux machines,
- le deuxième est celui des contraintes sur changement d'état et il en distingue deux catégories.

1 Contrainte procédurale (CPRO), pour la prise en compte de protocoles de marche à respecter entre machines,

2 Contrainte d'observabilité (CO), pour exprimer la possibilité d'accéder aux informations nécessaires à la commande d'une machine,

La prise en compte des contraintes consiste donc à déterminer, à partir des liens entre machines, les différents états dans lesquels se trouvent ces machines lors du fonctionnement. Cette démarche aboutit à un modèle structuro-fonctionnel permettant de spécifier ce type de contraintes. Le seul problème que nous constatons est que le modèle proposé suppose que les différents liens qui existent entre les machines d'un système sont statiques et ne peuvent pas varier lors du fonctionnement ou du cycle de vie. En effet si ce cas se présente pour un système donné, il faudra spécifier autant de modèles structuro-fonctionnels qu'il y a de changements possibles dans les contraintes comme elles sont définies par [BOIS 91]. On risque donc de se trouver avec un très grand nombre de modèles structuro-fonctionnels pour spécifier les contraintes d'un même système.

4.2.3. Notion de contrainte pour [ELKA 93]

La définition donnée par [ELKA 93] à la notion de contrainte est la suivante :

"les contraintes se formulent par l'interdiction de réaliser une action d'un composant quand un autre composant se trouve dans un état particulier".

L'approche de [ELKA 93] part de l'hypothèse que chaque composant élémentaire d'un système est caractérisé par un ensemble d'états physiques et un ensemble d'actions qu'il réalise. La notion de contrainte porte sur les restrictions qui limitent les actions à entreprendre ceci en fonction des états des différents composants d'un système global. La démarche utilisée est une démarche structurelle organisant dans un tableau les différents liens. Cette démarche nous a posé quelques problèmes pour spécifier des liens dynamiques du type : une action "A" ne peut être réalisée que si un composant est dans un état "E" depuis plus de cinq secondes.

4.2.4. Notion de contrainte pour [VEGA 96]

C'est dans les travaux de [VEGA 96] que nous avons trouvé une approche qui introduit tout d'abord la notion de contrainte d'une manière générale. Ainsi, pour [VEGA 96] une

contrainte est une restriction à l'un des attributs (une caractéristique) d'un système. [VEGA 96] s'intéresse au cas particulier des caractéristiques temporelles des applications temps réel. Il établit un lien entre la notion de contrainte et celle d'événement afin de dissocier la formulation d'une contrainte de celle des instants ou cette contrainte est vérifiée. Ainsi, la notion de contrainte temporelle pour [VEGA 96] consiste à formuler les relations de dépendance qui existent entre les occurrences des événements.

4.2.5. Notre contexte d'utilisation de la notion de contrainte

Dans notre démarche d'expression des contraintes, nous nous plaçons dans le contexte où la conduite des services d'un ensemble d'instruments intelligents a pour objectif la réalisation d'une application particulière. Dans ces conditions, les requêtes de demande d'exécution de service peuvent se trouver restreintes par des règles et des comportements propres à l'application souhaitée. Dès lors, les contraintes sont des restrictions qui portent sur les choix possibles (de requêtes à émettre) que peuvent faire les utilisateurs. Nous appelons ce type de contraintes : contraintes entre requêtes.

Par ailleurs, les origines des restrictions de conduite qui définissent les contraintes ne nous intéressent pas. Ces restrictions peuvent être dues : à des séquences d'exécutions obligatoires ; à des procédures de démarrage ou d'arrêt imposées ; à des performances souhaitées ; à des liens entre machines [BOIS 91] ; à des liens entre les actions et les états des machines [ELKA 93] ; etc. Nous postulons simplement que ces restrictions peuvent exister et nous étudions cette éventualité.

4.3. Spécification de la conduite d'une application : expression des contraintes dans un graphe de menus d'utilisations.

4.3.1. Position du problème

L'existence de contraintes entre requêtes relatives à l'ensemble des services d'une application pose deux problèmes. Le premier est de formaliser l'existence d'une contrainte. Le second est d'exprimer la répercussion globale, au niveau de l'utilisateur, de toutes les contraintes qui peuvent exister. Il faut donc disposer d'un outil formel pour exprimer comment les contraintes limitent le choix (de services à exécuter) que peut faire l'utilisateur.

4.3.2. Activabilité d'un service

L'absence de contraintes sur l'ensemble des services d'une application se traduit pour l'utilisateur par la possibilité d'exécuter à tout moment le service de son choix : tous les services disponibles sont activables à n'importe quel moment. En fait bien que toutes les ressources nécessaires à l'exécution du service soit disponibles, il peut exister des conditions à vérifier avant que ce dernier ne soit activable par l'utilisateur. Par exemple, une procédure quelconque peut imposer qu'un service soit exécuté après un autre. Ce sont ces conditions que nous appelons contraintes et qui font que l'utilisateur n'est pas totalement libre dans ses choix.

Exemples :

- le tapis A ne peut être mis en rotation que 5 secondes après le tapis B [chp 6]
- 10 secondes après il est possible de mettre en marche les extracteurs E1 et E2 [chp 6]
- ...

Définition 4.1:

On dit qu'un service est activable à un instant donné (dans le contexte d'une application), si son exécution peut être demandée à cet instant.
On appelle activabilité d'un service (s) une variable binaire qu'on note A(s) telle que A(s)=1 si et seulement si le service (s) est activable.

Remarques:

1- Une requête de l'utilisateur fait passer un service de l'état Activable à l'état Actif (demandé).
2- Si le service n'est pas activable la requête est rejetée.
3- Si l'on souhaite expliciter le caractère dynamique de A(s) on peut noter A(s,t) la valeur de A(s) à l'instant t

D'un point de vue externe, une contrainte sur un service se traduit par l'existence d'une condition à vérifier pour que le service soit activable par l'utilisateur. C'est-à-dire qu'il existe une assertion dont la valeur de vérité détermine si le service est activable, ou non. Ainsi, le service est activable si l'assertion conditionnant son activabilité est vraie, il est inactivable par l'utilisateur dans le cas contraire. Ces assertions peuvent faire intervenir n'importe quel

type de variables et expriment la répercussion au niveau de l'utilisateur des spécifications de conduite relatives à l'installation globale.

Conclusion :

Une contrainte sur un service se décrit par une assertion dont la valeur de vérité conditionne, à chaque instant, l'activabilité du service par l'utilisateur.

4.3.3. Comment exprimer les contraintes ?

Une contrainte est propre à une application. L'ensemble des contraintes d'une application est l'ensemble de toutes les assertions conditionnant l'activabilité des différents services offerts à l'utilisateur. Exprimer une contrainte revient à exprimer l'assertion qui la décrit.

Il existe plusieurs outils pour exprimer une assertion, on peut noter : la logique combinatoire, la logique spatio-temporelle [WIEC 94] ou tout autre outil permettant d'exprimer une condition ou une clause précise. Dans notre cas nous ne cherchons pas à définir ou à choisir un modèle particulier pour exprimer ces assertions, seuls leur existence et leurs effets sur la conduite nous préoccupent. Pour une entité externe l'important est de savoir à tout moment quels services sont activables et quels sont ceux qui ne le sont pas.

Sur une échelle des temps préalablement choisie [§4.3.5], l'évaluation, **à un instant donné,** d'une assertion exprimant une contrainte sur un service, peut produire du point vue de l'utilisateur trois conséquences possibles :

- 1 - la valeur de vérité de l'assertion passe de zéro à un, auquel cas le service devient et reste activable par l'utilisateur jusqu'à ce que l'événement 2 se produise.
- 2 - la valeur de vérité de l'assertion passe de un à zéro, auquel cas le service devient et reste inactivable par l'utilisateur jusqu'à ce que l'événement 1 se produise.
- 3 - la valeur de vérité de l'assertion reste inchangée, auquel cas l'état d'activabilité du service ne change pas.

D'une façon plus générale, une assertion peut conditionner plusieurs services à la fois.

Définition 4.2 :

On dit qu'un événement, conditionnant des services, s'est produit à **un instant** donné, si la valeur de vérité d'une des assertions exprimant les contraintes change à **cet instant.**

Un événement peut donc être caractérisé par :

- S^+= {l'ensemble des services qui deviennent activables par l'utilisateur à l'occurrence de l'événement}.
- S^-= {l'ensemble des services qui deviennent inactivables par l'utilisateur à l'occurrence de l'événement}.

Critère de cohérence

On peut dégager un premier **critère de cohérence** d'une assertion à savoir : $S^+ \cap S^- = \varnothing$.

En conclusion, une entité externe voit l'ensemble des services qu'elle peut activer invariable, jusqu'au moment où l'évaluation d'une assertion impose une modification de l'ensemble des services activables. Dans ce cas, on dit qu'un événement conditionnant des services s'est produit. Ainsi, l'utilisateur voit l'ensemble des services qu'il peut activer invariable jusqu'à l'occurrence d'un événement conditionnant des services (voir figure 4.1 [§4.3.6]).

L'énumération de l'ensemble des événements et des services qu'ils conditionnent permettra dans la suite d'exprimer l'effet des contraintes au niveau de l'utilisateur. Par ailleurs, notons pour le moment qu'une analyse bibliographique sur le concept d'événement a été réalisée par [VEGA 96]. Cette analyse souligne que ce concept est relativement naturel pour la description des systèmes réactifs et des applications réparties. D'après [VEGA 96] un événement est représenté par une condition logique (donc une assertion). Cette dernière définition est parfaitement en accord avec notre interprétation de la notion d'événement. Dans ses travaux [VEGA 96] souligne bien que la notion d'occurrence d'événement permet de séparer nettement la condition logique de l'instant auquel elle se produit, en effet un événement est unique par contre ses occurrences peuvent être multiples.

Conclusion

Une contrainte se traduit du point de vue externe par l'existence d'événements dont les occurrences conditionnent l'activabilité, par l'utilisateur, de certains services.

4.3.4. Structure de l'ensemble des contraintes

Le nombre d'assertions qui expriment les contraintes sur l'ensemble des services d'une application peut théoriquement être infini. Cependant les conséquences de ces assertions, c'est-à-dire les types d'événements que peuvent générer ces assertions, sont limités par le nombre de couples (S^+ , S^-) introduits dans la définition 4.2 et qui caractérisent chaque événement.

Proposition 4.1:

Le nombre d'événements exprimant les contraintes sur un ensemble de service est fini.

Démonstration

Il est facile de voir que, du point de vue de l'utilisateur, le nombre d'événements différents qui peuvent se produire est strictement inférieur à $2^{(nb\_services)} \times 2^{(nb\_services)} = 2^{2(nb\_services)}$ (On peut démontrer que le nombre maximum d'événements possibles est $3^{nb\_services}$, il est égal au nombre de couples (S^+;S^-) tels que $S^+ \cap S^- = \varnothing$; Démonstration Annexe 1). Il est donc possible de conclure que le nombre d'événements vu par l'utilisateur est fini.

Définition 4.3:

On définit une application par un couple formé d'un ensemble de services et d'un ensemble d'événements conditionnant ces services : $\mathcal{A} = (S(\mathcal{A})=\{s_i\} ; E(\mathcal{A})=\{e_j\})$

Définition 4.3 bis:

On appelle ensemble de contraintes d'une application "$\mathcal{A}$", on notera $E(\mathcal{A})$, l'ensemble de tous les événements conditionnant l'activabilité par l'utilisateur des différents services.

Le problème est de formaliser la relation qui lie l'occurrence d'un événement à la possibilité d'activation (ou pas) du service par l'utilisateur. D'après la définition 4.2, ces relations sont du type:

Le service devient activable **dès que** l'événement se produit

ou Le service devient inactivable **dès que** l'événement se produit

Ce type de relations lie l'état d'une variable aux instants d'occurrence d'un événement. La formulation de ces relations est typiquement un problème de logique temporelle.

Un problème de logique temporelle se définit dans un modèle dit modèle d'observateur temporel externe [WIEC 94] ou modèle temporel. Dans ce modèle, les relations du type cité plus haut se décrivent par des assertions ou clauses. Les assertions sont des formules temporelles construites à partir de variables logiques auxquelles on applique les opérateurs booléens classiques et des opérateurs temporels.

4.3.5. Modèle d'observateur temporel externe [WIEC 94]

La logique temporelle est une méthodologie formelle pour la spécification et le développement des systèmes temps réel [WIEC 94]. Elle permet de décrire des propriétés sur le temps qui doivent être vérifiées lors du fonctionnement du système [WEIC 94]. Elle se formalise à l'aide d'assertions dans un modèle d'observateur temporel [SHAR 95]. Ce modèle définit une projection de l'ensemble des occurrences des événements sur une échelle des temps prédéfinie dans le modèle . Cette notion de modèle temporel a été très bien décrite dans les travaux de M.J.Wieczorek [WIEC 94], ce dernier a même étendu ce modèle pour lui donner une dimension spatiale. Il définit un modèle spatio-temporel pour spécifier les systèmes répartis temps réel.

La suite de ce paragraphe est en partie un rappel des notions de base du modèle temporel. Ce dernier est essentiellement construit à partir des notions suivantes :

Une échelle des temps

Avant toute considération temporelle il faut d'abord définir le temps. Le temps est vu comme concept mais également comme une variable [SHAR 95], cette variable peut être réelle (temps continu), entière (temps discret) ... Il est donc important de préciser la topologie du temps. M.J.Wieczorek propose [WIEC 94] plusieurs topologies possibles, voire une topologie circulaire. Il retiendra une topologie identique à celle de l'ensemble des réels. C'est également celle que nous retiendrons pour la suite.

Une métrique temporelle

Elle se définit comme n'importe quelle métrique sur un espace vectoriel. Une métrique temporelle permet d'évaluer les distances entre les instants d'occurrence des événements. Avec la topologie du temps choisie, notre domaine de métrique temporelle est l'ensemble des réels positifs.

Un langage de formulation

C'est un langage qui permet une quantification dans le domaine métrique, c'est-à-dire qu'il relie les occurrences des événements entre elles par leurs positions relatives dans l'échelle des temps. Ce langage s'exprime à l'aide des outils suivants :

1- Un ensemble de symboles de variables, E, dans notre cas c'est l'ensemble des événements conditionnant les services.

2- Des symboles opérationnels opérant sur le domaine métrique : + ; = ; <

3- Les opérateurs booléens classiques : $\neg$; $\wedge$; $\vee$; $\rightarrow$

4- Les connectives temporelles suivantes : $\Box$; ∇ ; Δ ; $\Diamond$;

Elles opèrent sur les projections de E sur l'échelle des temps et ont les significations suivantes : Soit **e** $\in$ E et 0 l'instant courant (de référence)

$\boxplus$ **e** $\Leftrightarrow \forall t > 0$ **e**(t) = 1 (**e** devient vrai) ; $\boxplus$**e** $\Leftrightarrow \forall t \geq 0$ **e**(t) = 1 ;

$\boxminus$ **e** $\Leftrightarrow \forall t < 0$ **e**(t) = 1 (**e** était vrai) ; $\boxminus$ **e** $\Leftrightarrow \forall t \leq 0$ **e**(t) = 1 ;

$\Diamond$ **e** $\Leftrightarrow \neg \boxplus \neg$ **e** $\Leftrightarrow \exists t > 0$ tel que **e**(t)= 1 (e sera vrai);

$\Diamond$ **e** $\Leftrightarrow \neg \boxplus \neg$ **e** $\Leftrightarrow \exists t \geq 0$ tel que **e**(t)= 1 (e a été vrai)

$\Diamond$ **e** $\Leftrightarrow \neg \boxminus \neg$ **e** ; $\Diamond$ **e** $\Leftrightarrow \neg \boxminus \neg$ **e**;

Δ**e** $\Leftrightarrow \forall t \neq 0$ **e** =1 ; Δ **e** $\Leftrightarrow \forall t$ **e** =1 (e est toujours vrai)

∇**e** $\Leftrightarrow \neg \Delta \neg$**e** (e est parfois vrai)

Conclusion

Dans ce formalisme les deux relations qui peuvent lier l'occurrence des événements à l'activabilité d'un service [§4.3.2] s'écrivent :

$e_1 \rightarrow \boxplus$ A(s) $\Leftrightarrow$ dés que $\mathbf{e_1}$ se produit **s** devient activable

ou $e_0 \rightarrow \boxplus \neg$A(s) $\Leftrightarrow$ dés que $\mathbf{e_0}$ se produit **s** n'est plus activable

4.3.6. Effet de l'ensemble des contraintes sur la conduite

Notre préoccupation est de savoir à tout moment quels services sont activables et quels sont ceux qui ne le sont pas. Pour ceci il nous faut des repères dans le temps marquant les débuts

et les fins d'activabilité des différents services. Afin d'éviter d'introduire de nouvelles notions, il peut être judicieux d'utiliser les instants d'occurrences des événements pour se repérer dans le temps, ceci permet [§ 4.3.7] de définir les étapes de fonctionnement ainsi que l'état du système lors du fonctionnement. Nous obtenons ainsi des chronogrammes du type de la figure 4.1.

L'intérêt est d'éviter tout autre système de repérage dans le temps, aussi bien une mesure (relative ou absolue) du temps que la formulation de relations compliquées.

Cette démarche n'est possible que grâce au fait que la relation qui lie l'occurrence des événements à l'activabilité d'un service est de l'une des deux formes suivantes [§4.3.5] :

$\boxplus\ e_1 \rightarrow A(s)$

ou $\boxplus\ e_0 \rightarrow \neg A(s)$

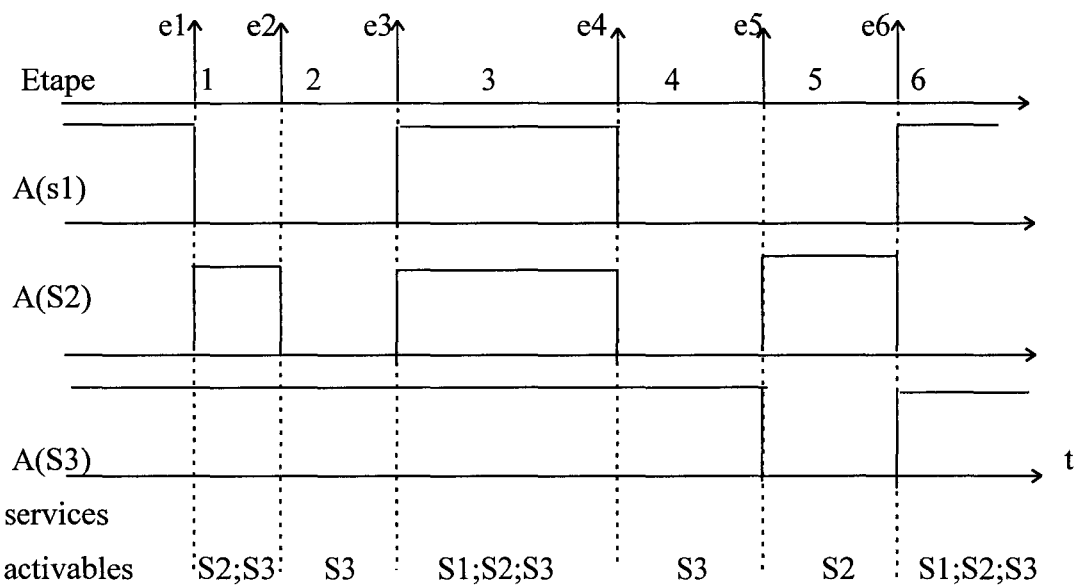


Figure 4.1 : Effet des contraintes sur la conduite (exemple)

4.3.7. Formulation du problème des contraintes.

Définition 4.4:

On dit qu'il existe une contrainte sur un service (s) si et seulement si il existe un événement (e) pouvant se produire tel que l'une des deux propositions suivantes est vraie:

(1) $e \rightarrow \boxplus A(s)$

(2) $e \rightarrow \boxplus \neg A(s)$

Remarque :

Le même événement peut conditionner plusieurs services de façons différentes [Déf 4.2].

Définition 4.5:

A l'instant T_k l'état du système est défini par la séquence temporisée de tous les événements qui se sont produits à des instants $T_i \leq T_k$, depuis une origine donnée T_0.

$X(T_k) = [e(T_1); e(T_2); ...; e(Tk)]$.

Remarques :

1- Le choix de la topologie du temps fait dans [§4.3.5] est une topologie dense, c'est-à-dire que la granularité du temps est suffisamment faible ce qui permet de supposer que deux événements ne se produisent pas en même temps.

2- On suppose qu'il existe une date T_0 avant laquelle le système n'existe pas, c'est-à-dire que le système n'a pas d'historique avant T_0.

Définition 4.6:

On appelle étape de fonctionnement d'une application tout intervalle de temps [Tk;Tk+1[pendant lequel l'état du système n'a pas changé c'est-à-dire aucun événement de $E(\mathcal{A})$ ne s'est produit.

$\forall t \in [Tk;Tk+1[, X(t) = X(Tk)$

Cet intervalle de temps est donc encadré par l'occurrence de deux événements successifs.

Proposition 4.2:

Etudions toutes les étapes de fonctionnement possibles.

Il existe pour chaque service d'une application au moins une étape possible pendant laquelle le service est activable par l'utilisateur.

Démonstration : par l'absurde

soit un service tel que : $A(s) = 0 \ \forall$ Tk et $\forall$ X(Tk)

Ainsi, le service n'est jamais activable, il ne peut jamais être demandé par les utilisateurs, c'est-à-dire que du point de vue des utilisateurs ce service n'existe pas donc n'appartient pas à l'application. S'il en est autrement, le système est mal conçu.

D'après la proposition 4.2, on peut dégager un critère de conception cohérente d'une application. En effet il doit exister pour chaque service proposé une étape possible pendant laquelle le service est activable par les utilisateurs. Ce critère se formalise de la façon suivante:

Critère de conception cohérente

Soit S($\mathcal{A}$) l'ensemble des services nécessaire pour la réalisation d'une application "$\mathcal{A}$".

$\forall s \in S(\mathcal{A}) \exists T_k$, T_{k+1} et $X[T_k]$ tel que $\forall t \in [T_k , T_{k+1}[$ $A(s,t) = 1$

On peut aussi formuler ce critère en logique temporelle :

$\forall s \in S(\mathcal{A}) \exists ei \in E(\mathcal{A})$ tel que, la propositions suivante est vraie:

$$e_i \rightarrow \boxdot A(s)$$

A ce stade nous savons formaliser du point de vue de la commande les états et les étapes de fonctionnement d'une application. Il faut maintenant formuler les conséquences sur les choix des utilisateurs.

Proposition 4.3:

Durant une étape de fonctionnement l'ensemble des services activables (c'est-à-dire disponibles sans contraintes) est invariable.

Démonstration : par l'absurde

soit (s) un service tel que A(s) change d'état pendant une étape de fonctionnement $\Leftrightarrow$ il s'est produit pendant l'étape un événement conditionnant le service

Ceci est impossible, car une étape de fonctionnement est un intervalle de temps pendant lequel aucun événement de E($\mathcal{A}$) ne s'est produit, en effet l'occurrence d'un événement marquerait le début d'une nouvelle étape.

Définition 4.7:

On appelle menu d'une application associé à une étape de fonctionnement l'ensemble des services activables que propose l'application pendant cette étape. On note $\mu[X(Tk)]$ (ou μ_k s'il n'y a pas de confusion possible)

Remarques:

1- Le menu d'une étape peut être vide ; dans ce cas, le système est dit non commandable par l'utilisateur durant cette étape.

2- Des menus identiques peuvent être associés à des étapes différentes.

Définition 4.8:

On note $E[X(T_k)]$ l'ensemble des événements de E($\mathcal{A}$) qui peuvent se produire quand le système est dans l'état $X(T_k)$. (on notera E_k s'il n'y a pas de confusion possible)

Proposition 4.4:

Connaissant le menu d'une étape $\mu[X(T_k)]$ et l'ensemble des événements possibles durant cette étape E[X(Tk)], on peut déterminer tous les menus $\mu[X(Tk+1)]$ possibles.

Démonstration :

Soit $X(T_k) = [e(T_1);e(T_2); \ldots ; e(T_k)]$

Soit $X(T_{k+1}) = [e(T_1);e(T_2); \ldots ; e(T_k);e(T_{k+1})]$

D'après la définition 4.2 on a :

$S^+[e(T_{k+1})]$ l'ensemble des services qui deviennent activables quand $e(T_{k+1})$ se produit

$S^-[e(T_{k+1})]$ l'ensemble des services qui deviennent non activables quand $e(T_{k+1})$ se produit

Le menu $\mu[X(T_{k+1})]$ est déduit de $\mu[X(T_k)]$ en ajoutant les services qui deviennent activables et en retranchant ceux qui deviennent non activables à l'occurrence de $e(T_{k+1}) \in E[X(T_k)]$.

$$\mu[X(T_{k+1})] = \left[\mu[X(T_k)] \cup S^+[e(T_{k+1})]\right] - S^-[e(T_{k+1})].$$

avec

$e(T_{k+1}) \in E[X(T_k)]$.

Conclusion :

Le triplet $\left[\mu[X(T_k)] ; e(T_{k+1}) ; \mu[X(T_{k+1})]\right]$ est unique c'est-à-dire que connaissant le menu source et l'événement qui s'est produit il existe un unique menu destination.

4.3.8. Graphe des Menus ou Graphe des contraintes

Notre but est de montrer qu'on peut exprimer les contraintes relatives à une application dans un graphe états transitions, ce dernier détermine les conditions de passage d'un menu à l'autre.

Proposition 4.5:

Il est possible d'exprimer les contraintes relatives à une application en représentant les menus et les événements dans un graphe états transitions.

Démonstration.

D'après la proposition 4.4 les différents menus (μ_k) d'une application et les événements conditionnant les différents services peuvent être représentés par le système (1) suivant:

$$\mu_{k+1} = [\mu_k \cup S^+(e_{k+1})] - S^-(e_{k+1})$$

(1) avec

$$e_{k+1} \in E_k \subseteq E(\mathcal{A})$$

Au lieu de ne considérer, à chaque étape, que les événements qui peuvent se produire, on peut étudier le cas englobant où, à chaque étape, tous les événements ont la possibilité de se produire. On obtient le système (2) suivant :

$$\mu_{k+1} = [\mu_k \cup S^+(e_{k+1})] - S^-(e_{k+1})$$

(2) avec

$$e_{k+1} \in E(\mathcal{A})$$

Ainsi à une étape donnée la simple connaissance de μ_k suffit pour déterminer les μ_{k+1} possibles (en effet d'après la proposition 4 chaque événement de $E(\mathcal{A})$ nous conduit à un unique menu μ_{k+1}) le système (2) définit donc un processus de Markov. Pour un utilisateur extérieur l'état du système, à une étape donnée, est entièrement déterminé par le menu μ_k de l'étape. Pour connaître le menu à proposer à l'utilisateur, il n'est plus nécessaire de définir l'état du système par la séquence de tous les événements qui se sont produits, la simple donnée de μ_k suffit. Le système à donc un nombre d'états (nb_états) tel que :

nb_états $\leq |\, P(S) \,|$ (cardinal de l'ensemble des parties de S)

nb_états $\leq 2^{nb\_service}$ avec (nb_service) = Card(S).

En Résumé

1- Le nombre d'états est fini (chaque menu représente un état).

2- Pour chaque état on connaît les événements qui peuvent se produire, chaque événement conditionne un arc sortant.

3- Pour chaque arc sortant on connaît l'état destination celui-ci est unique [Proposition 4.4].

Conclusion

On peut représenter les menus et les événements dans le graphe états transitions suivant :

$G_\mu(P(S) ; T)$ avec :

P(S) l'ensemble des menus inclus dans l'ensemble des parties de S

et $T = \{ (\mu_i ; e_j ; \mu_k) \in P(S) \times E(\mathcal{A}) \times P(S)$ tel que $\mu_k = [\mu_i \cup S^+(e_j)] - S^-(e_j) \}$

Le graphe G_μ représente le cas le plus exhaustif où l'on prévoit toutes les possibilités envisageables c'est-à-dire qu'on étudie, à toutes les étapes possibles, les conséquences de l'occurrence de chaque événement.

Le système (1) est un sous système de (2), il correspond aux cas où certains événements ne peuvent pas se produire à certaines étapes de fonctionnement ou aux cas où certains événements ne peuvent se produire que si d'autres se sont produits. La résolution du problème revient dans le premier cas à éliminer certaines transitions correspondantes dans le graphe G_μ, dans le deuxième cas il suffit de conditionner les transitions correspondantes dans le graphe G_μ en plus des événements de tir par les conditions sur les événements en question.

Ainsi le système (1) correspond à un sous graphe du graphe G_μ .

Définition 4.9:

On appelle indifféremment graphe des menus ou graphe des contraintes le graphe exprimant les contraintes relatives à une application.

Conclusion

L'existence de contraintes pour une application se traduit par l'existence d'événements conditionnant les activabilités des différents services. Ces contraintes s'expriment par des assertions liant l'occurrence des événements aux activabilités des services. Ces assertions définissent une chronologie qui gère la façon de proposer les services à l'utilisateur, ainsi à chaque étape l'application propose un ensemble de services dit menu d'utilisation. Les menus et les événements peuvent être représentés dans un graphe état transitions. Les sommets du graphe sont les menus. Dans un menu, l'utilisateur est libre d'exécuter le service de son choix. Quant aux transitions, elles représentent les possibilités de passage d'un menu à l'autre et sont conditionnées par les événements susceptibles de modifier l'ensemble des services activables.

4.3.9. Méthode de construction du graphe des menus

Dans ce qui précède [Proposition 4.5], le graphe états transitions permettant d'exprimer les contraintes relatives à une application est déduit d'un graphe plus général Gμ [Proposition 4.5 conclusion]. La construction de Gμ suppose le recensement de tous les événements qui peuvent conditionner un service à une étape donnée.

Toutefois, il est possible d'obtenir le graphe des menus par une procédure récurrente [ch6 §6.6.4]. L'idée est basée sur la proposition 4.4: il s'agit de construire le graphe

progressivement en partant d'un menu initial μ_0. La démarche consiste, tout d'abord, à trouver pour l'étape courante tous les événements qui peuvent se produire, chacun de ces événements peut marquer le début d'une des éventuelles étapes qui peuvent succéder à l'étape courante [Définition 4.5 et 4.6]. La suite de la démarche consiste à déterminer [proposition 4.4] pour chacune de ces nouvelles étapes le menu qui lui correspond. De cette façon on obtient les transitions du graphe qui sortent du menu courant ainsi que le menu destination de chaque transition. Pour chacune de ces nouvelles étapes il faut reprendre la procédure, celle-ci s'arrête chaque fois qu'on se retrouve à une étape qui existe déjà dans le graphe. Cette démarche est détaillée dans le cas de l'exemple traité dans [ch6].

4.4. Simplification de la conduite d'une application : partition de l'ensemble des menus en classe d'équivalences .

4.4.1. Position du problème.

Le graphe des menus d'une application permet de présenter, à chaque étape de fonctionnement, l'ensemble des services activables. L'existence de ce graphe nécessite la mise en œuvre d'un mécanisme de gestion des menus lors du fonctionnement. La méthode de construction du graphe des menus introduite dans [§4.3.9] n'interdit pas l'obtention d'un graphe dans lequel des sommets différents proposent le même menu [voir figure 6.3]. Ce type de situations correspond à des étapes de fonctionnement différentes qui proposent le même menu. Deux étapes proposent le même menu si les deux événements (e_i et e_j), marquant chacun par son occurrence le début d'une des étapes, vérifient l'égalité suivante:

$$[\mu_{i-1} \cup S^+(e_i)]\text{-}S^-(e_i) = [\mu_{j-1} \cup S^+(e_j)]\text{-}S^-(e_j)$$

Afin de simplifier la gestion du graphe des menus on se propose de fusionner les sommets du graphe qui proposent le même menu. L'avantage réside surtout dans le fait que le nombre de sommets se trouve ainsi limité par une valeur maximale connue ($2^{nb\_services}$).

L'exemple représenté figure 4.2 exprime bien l'objectif de la démarche. Il est évident que pour l'utilisateur il y a une équivalence totale entre les deux graphes. En effet quel que soit l'événement qui se produit, l'utilisateur se voit proposer dans les deux cas le même menu de services activables. Cependant, en termes de mise en place des mécanismes de leur gestion, ces deux graphes ne sont pas équivalents. Le premier met en œuvre des transitions dont

l'évaluation est simple, au prix d'un nombre de sommets important, dont certains représentent en fait le même menu. Le second réduit cette redondance, mais son plus petit nombre de sommets s'obtient au prix d'une complexification de ses transitions. Nous étudions dans ce qui suit le passage d'un graphe des menus au graphe de ses classes d'équivalence.

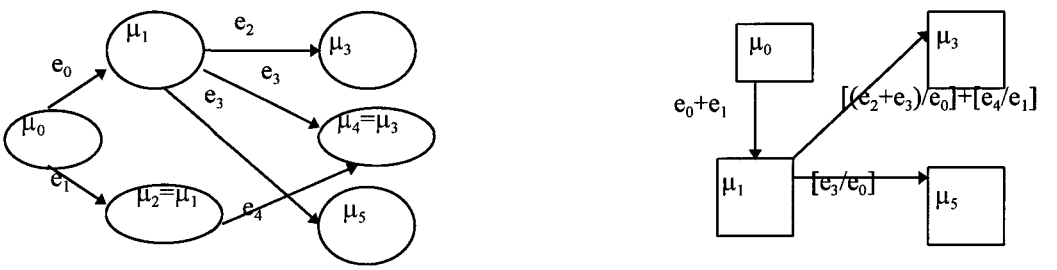


Figure 4.2 : Exemple
(La notation $[e_i/e_j]$ signifie : l'occurrence de e_i à condition que e_j se soit produit auparavant)

4.4.2. Formulation du problème

4.4.2.1. Rappel [BILA 94] [ARNO 92]

Définition 4.10 :

Soit G = (S ; T) un système états transitions et soit D un sous ensemble de S, on définit :

l'ensemble des transitions ayant leur extrémité initiale dans D et terminale dans $\underline{D}$=S-D

$w^+_G(D) = \{t_i \in T / \alpha(t_i) \in D \text{ et } \beta(t_i) \notin D\} = \{(x,y,z) \in S \times A \times S / x \in D \text{ et } y \notin D\}$

l'ensemble des transitions ayant leur extrémité terminale dans D et initiale dans $\underline{D}$=S-D

$w^-_G(D) = \{t_i \in T / \beta(t_i) \in D \text{ et } \alpha(t_i) \notin D\} = \{(x,y,z) \in S \times A \times S / x \notin D \text{ et } y \in D\}$

On note $w_G(D) = w^+_G(D) \cup w^-_G(D)$

4.4.2.2. Agrégation des menus en classe d'équivalences.

Définition 4.12:

On dit que deux menus μ_i et μ_j, relatifs à deux sommets différents d'un graphe de menus, sont équivalents s'ils proposent le même ensemble de services activables.

On note :

$\mu_i \equiv \mu_j$ pour exprimer l'équivalence

et $\underline{\mu_i}$ pour représenter la classe d'équivalence du menu μ_i

Proposition 4.6:

Il est facile de voir que "≡" est une relation d'équivalence.

Cette équivalence reflète le fait que l'équipement propose à l'utilisateur le même ensemble de services activables, pour ce dernier il n'y a aucune différence entre deux menus équivalents. On peut ainsi partitionner l'ensemble des menus relatifs à des sommets d'un graphe de menus en classes d'équivalences. L'objectif est de regrouper l'ensemble des menus d'une même classe en un seul sommet dans le graphe des menus.

4.4.2.3. Regroupement des menus d'une même classe d'équivalence en un seul sommet dans le graphe des menus.

Soit un graphe de menus $G = (\{\mu_i / \mu_i \text{ menu d'une étape i}\} , T_G)$,

et soit $\underline{\mu_i}$ une classe d'équivalence de l'ensemble des menus $\{\mu_i / \mu_i \text{ menu d'une étape i}\}$.

Le problème revient à trouver le graphe G qui représente le même type de fonctionnement que G mais dans lequel les menus de la classe $\underline{\mu_i}$ sont représentés par un seul sommet. Dans ce cas, on peut confondre par abus de notation la classe $\underline{\mu_i}$ avec le sommet qui la représente dans G et avec l'ensemble des services qu'elle propose.

4.4.2.3.1. Détermination des transitions entrantes vers le nouveau sommet

Il est facile de voir, comme va le démontrer la proposition 4.7 , que le fait d'accéder dans le graphe G au sommet relatif à la classe $\underline{\mu_i}$ correspond au fait d'accéder dans le graphe G à l'un des menus de cette classe. Ceci permet de conclure qu'à chaque transition (du graphe G) entrante vers l'un des menus de la classe $\underline{\mu_i}$, lui correspond sans ambiguïté dans le graphe G une transition entrante vers cette même classe. En revanche, le cas des transitions sortantes est légèrement plus complexe.

Propositions 4.7:

Soit un graphe de menus $G = (\{\mu_i / \mu_i \text{ menu d'une étape i}\} , T_G)$.

Soit $\underline{\mu_i}$ une classe d'équivalence de l'ensemble des menus $\{\mu_i / \mu_i \text{ menu d'une étape i}\}$.

Soit G le graphe qui représente le même type de fonctionnement que G mais où les menus de la classe $\underline{\mu_i}$ sont représentés par un seul sommet.

Nous avons alors les trois propositions suivantes :

1- si $\exists\ \mu_i \in \underline{\mu_i}$ et $\exists\ \mu_{i-1} \notin \underline{\mu_i}$ /$T_i = (\mu_{i-1}, e_i, \mu_i) \in w^-_G(\mu_i) \Rightarrow T = (\mu_{i-1}, e_i, \underline{\mu_i}) \in w^-_G(\underline{\mu_i})$.

2- si $\exists\ \mu_{i-1} \notin \underline{\mu_i}$ /$T= (\mu_{i-1}, e_i, \underline{\mu_i}) \in w^-_G(\underline{\mu_i}) \Rightarrow \exists\ \mu_j \in \underline{\mu_i}$ /$T_j = (\mu_{i-1}, e_i, \mu_j) \in w^-_G(\mu_j)$.

3- $w^-_G(\underline{\mu_i}) = \{ T= (\mu_{i-1}, e_i, \underline{\mu_i}) \in w^-_G(\underline{\mu_i}) / \exists\ \mu_j \in \underline{\mu_i}$ et $T_j = (\mu_{i-1}, e_i, \mu_j) \in w^-_G(\mu_j) \}$.

Démonstration

1-

Soit Ti = (μi-1 , ei , μi) ∈ w-G(μi) ⇔ Pour G l'événement ei peut se produire à l'étape (i-1) et conduit à $\mu_i = [\mu_{i-1} \cup S^+(e_i)] - S^-(e_i) \Rightarrow$ Pour G l'événement e_i peut se produire à l'étape (i-1) et conduit à $\underline{\mu_i} = [\mu_{i-1} \cup S^+(e_i)] - S^-(e_i) \Leftrightarrow \exists\ T = (\mu_{i-1}, e_i, \underline{\mu_i}) \in w^-_G(\underline{\mu_i})$

2-

par l'absurde

Soit $T= (\mu_{i-1}, e_i, \underline{\mu_i}) \in w^-_G(\underline{\mu_i}) / \forall\ \mu_j \in \underline{\mu_i}\ \nexists\ T_j = (\mu_{i-1}, e_i, \mu_j) \in w^-_G(\mu_j) \Leftrightarrow$ Pour G soit l'événement e_i ne peut pas se produire à l'étape i-1 soit $\forall\ \mu_j \in \underline{\mu_i}\ \ \mu_j \neq [\mu_{i-1} \cup S^+(e_i)] - S^-(e_i) \Rightarrow$ $T= (\mu_{i-1}, e_i, \underline{\mu_i})$ n'existe pas.

3-

(1) et (2) ⇔ (3)

4.4.2.3.2. Détermination des transitions sortantes du nouveau sommet

L'ensemble des transitions entrantes vers $\underline{\mu_i}$ est déterminé grâce à la proposition 4.7. Il faut maintenant trouver l'ensemble des transitions sortantes. Ce problème est un peu plus complexe, cherchons d'abord à étudier des cas simples.

Considérons le cas où tous les menus d'une classe d'équivalence possèdent une transition sortante dans le graphe G avec le même événement de tir et le même menu destination. Dans ce cas, nous obtenons le résultat suivant :

Proposition 4.8:

Soit $T_i = (\mu_i, e_i, \mu_{i+1}) \in w^+_G(\mu_i) / \forall\ \mu_j \in \underline{\mu_i}\ \exists\ T_j = (\mu_j, e_i, \mu_{i+1}) \in w^+_G(\mu_j) \Rightarrow$

$T = (\underline{\mu_i}, e_i, \mu_{i+1}) \in w^+_G(\underline{\mu_i}) \Rightarrow$

$w^+_G(\underline{\mu_i}) \supseteq \{ T= (\underline{\mu_i}, e_i, \mu_{i+1}) \in w^+_G(\mu_i) / \forall\ \mu_j \in \underline{\mu_i}\ \exists\ T_j = (\mu_j, e_i, \mu_{i+1}) \in w^+_G(\mu_j) \}$

Démonstration

$\forall \mu_j \in \underline{\mu_i} \exists T_j = (\mu_j , e_i , \mu_{i+1}) \in w^+_G(\mu_j) \Leftrightarrow \forall$ j étape de G / $\mu_j \in \underline{\mu_i}$, l'événement e_i peut se produire et conduit à l'étape i+1 / $\mu_{i+1} = [\mu_j \cup S^+(e_i)] - S^-(e_i) \Leftrightarrow$ chaque fois qu'on est dans la classe $\underline{\mu_i}$ l'événement e_i peut se produire et conduit à l'étape i+1 / $\mu_{i+1} = [\underline{\mu_i} \cup S^+(e_i)] - S^-(e_i) \Leftrightarrow T = (\underline{\mu_i} , e_i , \mu_{i+1}) \in w^+_G(\mu_j)$.

Pour les transitions du graphe des menus qui vérifient l'hypothèse de la proposition 4.8 le problème ne se pose pas. Ce type de transition définit sans ambiguïté une transition dans le graphe des classes de menus. Considérons à présent les transitions qui ne vérifient pas cette propriété.

Position du problème :

Soit un graphe de menus $G = (\{\mu_i / \mu_i$ menu d'une étape i$\}, T_G)$.

Soit $\underline{\mu_i} = \{\mu_i , \mu_j\}$

Soit $\underline{G}$: le graphe qui représente le même type de fonctionnement que G mais où les menus de la classe $\underline{\mu_i}$ sont représentés par un seul sommet.

Soit $T_i = (\mu_i , e_i , \mu_{i+1}) \in w^+_G(\mu_i)$

Pour cette transition, deux cas de figure sont possibles:

1er cas	**2éme cas**
$\exists T_j = (\mu_j , e_i , \mu_{i+1}) \in w^+_G(\mu_j)$	Il n'existe pas $T_j = (\mu_j , e_i , \mu_{i+1}) \in w^+_G(\mu_j)$
$T = (\underline{\mu_i} , e_i , \mu_{i+1}) \in w^+_{\underline{G}}(\underline{\mu_i})$	$T = (\underline{\mu_i} , e_i , \mu_{i+1}) \notin w^+_{\underline{G}}(\underline{\mu_i})$ car T_j n'existe pas. Pourtant : $\exists$ une possibilité de passage de $\underline{\mu_i}$ à μ_{i+1}. Quelle est la transition? : $T = (\underline{\mu_i} ; ? ; \mu_{i+1})$

D'après la proposition 4.7 on sait comment accéder à la classe $\underline{\mu_i}$. Cependant, dès qu'on se trouve dans $\underline{\mu_i}$, on ne distingue plus quel menu de cette classe à permis d'y accéder. Il se pose donc un problème d'indéterminisme sur les destinations futures. En effet μ_i peut nous conduire à μ_{i+1}, par contre μ_j ne le permet pas. Si l'on ne mémorise pas le moyen d'accès à $\underline{\mu_i}$ (par μ_i ou par μ_j) on ne peut pas savoir si on peut aller à μ_{i+1} ou non.

4.4.2.3.2.1. Problème d'indéterminisme posé par l'agrégation des menus en classes d'équivalences (ce problème se pose uniquement pour les transitions sortantes).

Définition 4.11:

Soit un graphe de menus G = ($\{\mu_i / \mu_i$ menu d'une étape i$\}$, T_G).

Soit μ_i et μ_{i+1} des menus de G.

On dit qu'il existe un indéterminisme sur la transition $T_i = (\mu_i , e_i , \mu_{i+1}) \in w^+_G(\mu_i)$ si et seulement si, il existe $\mu_j \in \underline{\mu_i}$ tel que $T_j = (\mu_j , e_i , \mu_{i+1}) \notin w^+_G(\mu_j)$

On note :

T: une transition posant un indéterminisme

et T : une transition ne posant pas d'indéterminisme

4.4.2.3.2.2. Levée de l'indéterminisme et détermination des transitions sortantes

Définition 4.12:

1- Soit un graphe de menus G = ($\{\mu_i / \mu_i$ menu d'une étape i$\}$, T_G). Pour chaque $\mu_i \in \{\mu_i / \mu_i$ menu d'une étape i$\}$ on définit $\omega(\mu_i)$ une variable binaire égale à 1 quant on se trouve dans le menu μ_i du graphe G

2- On note $[e_i / \omega(\mu_i)]$ le fait que l'événement e_i se produit quant $\omega(\mu_i) = 1$

Remarque :

L'expression la plus générale de $\omega(\mu_i)$ en fonction des événements de G peut être obtenue grâce aux travaux de Gondran [GOND 79] sur l'algèbre des chemins de la théorie des graphes. Elle correspond à la forme la plus générale d'une séquence d'événements permettant (d'après le graphe des menus) de se trouver dans le menu μ_i [un rappel sera fait dans le chapitre 5].

Nous avons déterminé grâce à la proposition 4.7 l'ensemble des transitions, du graphe des classes de menu, entrantes vers un menu $\underline{\mu_i}$. Nous pouvons maintenant déterminer grâce à la proposition qui suit l'ensemble des transitions sortantes.

Proposition 4.9:

Soit un graphe de menus G = ($\{\mu_i / \mu_i$ menu d'une étape i$\}$, T_G).

Soit $\underline{\mu_i}$ une classe d'équivalence de $\{\mu_i / \mu_i$ menu d'une étape i$\}$.

1- $\exists\, \mu_{i+1} \notin \underline{\mu_i} / \boldsymbol{T_i} = (\mu_i , e_i , \mu_{i+1}) \in w^+_G(\mu_i) \Rightarrow T = (\underline{\mu_i} , [e_i / \omega(\mu_i)] , \mu_{i+1}) \in w^+_G(\underline{\mu_i})$.

2- $T = (\underline{\mu_i} , e_i , \mu_{i+1}) \in w^+_G(\underline{\mu_i}) \Rightarrow$ soit $\exists\, \mu_j \in \underline{\mu_i}$ et $\exists\, \boldsymbol{T_j} = (\mu_j , e_j , \mu_{i+1}) \in w^+_G(\mu_j)$ soit $\forall\, \mu_j \in \underline{\mu_i}$

$T_j = (\mu_j , e_i , \mu_{i+1}) \in w^+_G(\mu_j)$.

3- $w^+_G(\underline{\mu_i}) = \{ T = (\underline{\mu_i}, e_i, \mu_{i+1}) / \exists \mu_j \in \underline{\mu_i}$ et $\exists T_j = (\mu_j, e_j, \mu_{i+1}) \in w^+_G(\mu_j) \} \cup \{ T = (\underline{\mu_i}, e_i, \mu_{i+1}) / \forall \mu_j \in \underline{\mu_i}\ T_j = (\mu_j, e_i, \mu_{i+1}) \in w^+_G(\mu_j) \}$

Remarque :

- Dans (1) le cas où $\mu_{i+1} \in \underline{\mu_i}$ se traduit par $T = (\underline{\mu_i}, [e_i / \omega(\mu_i)], \underline{\mu_i})$ une transition bouclée sur $\underline{\mu_i}$. Cette transition est totalement transparente pour l'utilisateur, il est donc inutile pour ce dernier de la représenter dans G. En revanche T peut être utile lors de la définition des services composés, on note donc $w^0_G(\underline{\mu_i})$ l'ensemble de ces transitions.
- Dans (2) l'événement e_i peut être différent de e_j par contre ($e_i = 1 \Rightarrow e_j = 1$) et pas forcément l'inverse.

Démonstration

1-

Soit $T_i = (\mu_i, e_i, \mu_{i+1}) \in w^+_G(\mu_i) \Leftrightarrow$ on peut passer de μ_i à $\mu_{i+1} \Leftrightarrow$ si on se trouve dans la classe $\underline{\mu_i}$ et que $\omega(\mu_i)=1$ l'événement e_i peut se produire et conduit à μ_{i+1}

$\Rightarrow T = (\underline{\mu_i}, [e_i/\omega(\mu_i)], \mu_{i+1}) \in w^+_G(\underline{\mu_i})$.

2-

par l'absurde

Soit $\underline{\mu_i}$ telle que $\forall \mu_j \in \underline{\mu_i}\ T_j = (\mu_j, e_j, \mu_{i+1}) \notin w^+_G(\mu_j)$ et $\exists \mu_j \in \underline{\mu_i}\ T_j = (\mu_j, e_i, \mu_{i+1}) \notin w^+_G(\mu_j)$

$\Rightarrow$ il n'est prévu aucune possibilité de passage d'un des menus de la classe $\underline{\mu_i}$ vers le menu μ_{i+1}

$\Leftrightarrow T = (\underline{\mu_i}, e_i, \mu_{i+1}) \notin w^+_G(\underline{\mu_i})$

3- (1) et (2) $\Rightarrow$ (3)

4.4.2.3.3. Détermination du graphe regroupant en un seul sommet les menus d'une même classe d'équivalence.

Proposition 4.10:

Soit un graphe de menus $G = (\{\mu_i / \mu_i$ menu d'une étape i$\}, T_G)$.

Soit $\underline{\mu_i}$ une classe d'équivalence de $\{\mu_i / \mu_i$ menu d'une étape i$\}$

Soit $G = (S_G, T_G)$ le graphe qui représente le même type de fonctionnement que G mais où les menus de la classe $\underline{\mu_i}$ sont représentés par un seul sommet

On a :

$S_{\underline{G}} = \{\mu_k \notin \underline{\mu_i}\} \cup \{$ le sommet $\underline{\mu_i}\}$

$T_{\underline{G}} = \{ T = (\mu_k , e_{kl} , \mu_l) \in T_G / \mu_k \notin \underline{\mu_i}$ et $\mu_l \notin \underline{\mu_i}\} \cup w^+_{\underline{G}}(\underline{\mu_i}) \cup w^-_{\underline{G}}(\underline{\mu_i}) \cup w^0_{\underline{G}}(\underline{\mu_i})$

Démonstration

Il est facile de voir que $\underline{G}$ est déduit de G en remplaçant l'ensemble des transitions faisant intervenir des menus de la classe $\underline{\mu_i}$ par $w^+_{\underline{G}}(\underline{\mu_i}) \cup w^-_{\underline{G}}(\underline{\mu_i}) \cup w^0_{\underline{G}}(\underline{\mu_i})$. Les transitions entre les autres menus ne sont pas modifiées.

4.4.3. Définition du graphe des classes de menus

Le paragraphe précédent définit un graphe dans lequel les menus d'une même classe sont regroupés en un sommet unique. En faisant la même chose successivement avec chaque classe, on aboutit au graphe des classes de menus dans lequel chaque classe est représentée par un seul sommet. Ce graphe peut aussi être défini de façon précise grâce à la proposition 4.11.

Proposition 4.11:

Soit un graphe de menus $G = (\{\mu_i / \mu_i$ menu d'une étape i$\}, T_G)$.

Soit $\underline{G}(\underline{\mu_i}) = (S_{\underline{G}}, T_{\underline{G}})$ le graphe qui représente le même type de fonctionnement que G mais où les menus de la classe $\underline{\mu_i}$ sont représentés par un seul sommet.

Soit $\underline{G}$ $(\{\underline{\mu_i} / \underline{\mu_i}$ une classe$\}, T_{\underline{G}})$ le graphe des classes de menus, nous avons alors :

1- $w^+_{\underline{G}}(\underline{\mu_i}) = \{ T = (\underline{\mu_i} , e , \underline{\mu_j}) / T_j = (\underline{\mu_i} , e , \mu_j) \in w^+_{\underline{G}(\underline{\mu_i})}(\underline{\mu_i}) \}$

2- $w^-_{\underline{G}}(\underline{\mu_i}) = \{ T = (\underline{\mu_j} , e , \underline{\mu_i}) / T_j = (\underline{\mu_j} , e , \mu_i) \in w^+_{\underline{G}(\underline{\mu_j})}(\underline{\mu_j}) \}$

2- $T_{\underline{G}} = \cup_{\underline{\mu}} w^+_{\underline{G}}(\underline{\mu}) = \cup_{\underline{\mu}} w^-_{\underline{G}}(\underline{\mu})$

Démonstration :

(1) et (2) découlent facilement des propositions 4.8 et 4.9

(1) et (2)$\Rightarrow$ (3).

Récapitulatif.

Dans cette partie, nous allons résumer la démarche que nous avons suivie pour obtenir le graphe des classes de menus.

L'idée est de procéder par étapes en montrant que l'on peut regrouper les menus d'une même classe d'équivalence en un seul sommet dans le graphe des menus. Dans ce but, la proposition 4.7 établit qu'il est possible de déterminer les transitions entrante vers le nouveau sommet. En revanche, le cas des transitions sortantes est plus délicat et peut poser des problèmes d'indéterminisme. Pour cette raison, nous avons tout d'abord étudié le cas simple où les transitons sortantes du nouveau sommet ne posent aucun problème. La proposition 4.8 démontre que sous certaines conditions le regroupement des menus d'une même classe ne pose pas d'indéterminisme. Cette dernière proposition à aussi permis d'identifier les cas où des problèmes d'indéterminisme peuvent se poser [Déf 11]. La proposition 4.9 montre que l'indéterminisme sur une transition sortante peut être levé grâce à l'évaluation d'une variable qui permet d'identifier le menu origine.

Le graphe définitif représentant tous les menus d'une même classe par un seul sommet, est obtenu en appliquant la démarche précédente à toutes les classes l'une après l'autre [proposition 4.11].

4.5. Conclusion

La problématique générale de ce chapitre concerne la conduite d'un ensemble d'instruments intelligents pour réaliser une application particulière. En effet, nous avons constaté que la notion de mode d'utilisation, introduite dans le chapitre 2, n'est pas un filtre de commande suffisant pour la réalisation d'une application particulière.

Pour résoudre le problème nous avons postulé que l'existence de contraintes entre requêtes relatives à l'ensemble des services d'une application se traduit par l'existence d'événements conditionnant l'activabilité des services par les utilisateurs. Ce postulat nous nous a permis dans un premier temps d'établir un lien entre la notion de contrainte et celle d'événement, ensuite de constater que durant un intervalle de temps encadré par les instants d'occurrence de deux événements, l'ensemble des services activables par l'utilisateur ne varie pas. Nous avons appelé ce type d'intervalle de temps "étape de fonctionnement" et l'ensemble des services activables "menu d'utilisation".

Ainsi nous avons pu établir que les contraintes d'une application peuvent être exprimées par un graphe états transitions dont les sommets représentent des menus de services activables.

Pour prévoir qu'une application est directement réalisable par un ensemble d'instruments intelligents il faut alors que son graphe de menus puisse être projeté sur celui des modes d'utilisation de l'architecture support.

Enfin dans la dernière partie de ce chapitre, nous avons simplifié le graphe des menus d'une application en regroupant en un seul sommet les menus proposant le même ensemble de services. L'idée était de procéder par étape, il s'agit de regrouper l'ensemble des menus d'une même classe d'équivalence en un seul sommet dans le graphe, et ensuite de recommencer successivement pour les autre classes de menus.

Troisième partie
Automatisation de la conduite d'une application répartie construite en interconnectant des instrument intelligents

Chapitre 5. Automatisation de séquences

5.1. Introduction :

Dans le chapitre précédent, nous avons étudié la répercussion au niveau de l'utilisateur des restrictions de conduite que peut imposer la finalité d'une application répartie construite en interconnectant des instruments intelligents. Nous savons ainsi définir la conduite à adopter pour réaliser une application particulière par un ensemble d'instruments. Nous souhaitons maintenant étudier la possibilité de déléguer des séquences de conduite d'une application à un système d'automatisation. Notre objectif est de définir des niveaux de commande supérieurs permettant à l'utilisateur une conduite plus élaborée. Il s'agit donc d'une démarche d'analyse et de définition du procédé d'automatisation d'une application répartie interconnectant des instruments intelligents.

Dans ce chapitre, nous allons définir des services composés représentant des séquences d'exécution de services de base des différents instruments d'une application donnée. Ces services d'un niveau supérieur permettent une commande plus élaborée en automatisant des séquences précises d'exécution. Cette démarche d'automatisation nous pose essentiellement deux problèmes. Le premier est de formaliser par rapport aux notions déjà introduites la définition de ce que peut être un service composé. Le deuxième est de retrouver et d'exprimer tous les services composés que peut autoriser une application donnée.

Nous résoudrons les deux problèmes posés plus haut grâce à la théorie des langages et à l'algèbre des chemins de la théorie des graphes. Notre démarche part de la constatation que les séquences d'exécution de services autorisées par une application donnée sont celles qui permettent d'évoluer dans son graphe des menus, c'est-à-dire de se déplacer d'un menu à un autre selon les chemins possibles du graphe des menus. Il paraît donc évident que les éventuels services composés pouvant être définis correspondent à certains des chemins du

graphe des menus. D'autre part et grâce aux travaux de [GOND 79] sur l'algèbre des chemins dans un graphe, il nous sera possible de déterminer l'expression la plus générale des mots du langage représentant les chemins permettant de passer d'un menu à un autre. Dès lors nous disposerons de règles pour reconnaître les séquences d'exécution de services effectivement réalisables par l'ensemble des équipements d'une application donnée, ceci tout en respectant les contraintes de l'application considérée. L'établissement de ces règles permettra, dans la dernière partie de ce chapitre, de définir un langage de description pour la spécification des services composés. Ce langage permet dans le cas d'une automatisation de certaines séquences de spécifier correctement les tâches à exécuter et de savoir si celles-ci sont autorisées.

Toutefois, l'exécution d'un service composé suppose l'occurrence de tous les événements du chemin à parcourir. Pour cette raison avant de définir ou d'automatiser l'exécution de ce type de service il faut pouvoir affirmer l'occurrence de tous les événements nécessaires. Pour garantir le respect de cette dernière condition, nous éliminons du graphe des contraintes tous les événements dont l'occurrence pourrait ne pas se vérifier. De cette façon, nous obtenons une définition de services composés pouvant être automatisés.

L'organisation du présent chapitre est structurée de la façon suivante :

- Tout d'abord, nous allons définir à quoi correspond la démarche d'automatisation.
- Ensuite, nous rechercherons à définir le langage exprimant toutes les commandes que peut formuler un utilisateur pour la conduite d'un ensemble de services sur lequel il n'existe aucune contrainte.
- Dans une autre étape, nous déterminerons le sous-graphe des menus sur lequel l'utilisateur à un contrôle total,
- Enfin, nous définirons le langage des séquences de requêtes d'exécution de services qui peuvent être déléguées à un système d'automatisation.

5.2. Démarche d'automatisation : position du problème

5.2.1. L'automatisation vue par l'analyse décisionnelle

L'intérêt que nous portons à la démarche de modélisation par analyse décisionnelle réside dans les travaux effectués [BUCK 94] pour formaliser le concept d'automatisation d'un processus. Bien que ces travaux restent (comme nous le verrons) incomplets et ne permettent pas une spécification précise du procédé d'automatisation, ils peuvent néanmoins nous servir de base pour une formulation rigoureuse de la démarche d'automatisation.

Avant de définir l'automatisation au sens de l'analyse décisionnelle des systèmes, rappelons que celle-ci s'inscrit dans la lignée des conceptions orientées objet pour la spécification de la conduite d'un système. En effet, l'analyse décisionnelle qualifie un équipement comme un ensemble d'objets immergés dans un environnement de processus [BUCK 94]. Dans ce sens, l'environnement est chargé de diriger et de contrôler ce processus par rapport à une finalité dont il est porteur et ceci par les actions qu'il entreprend. Vu du processus, l'environnement n'a aucune forme matérielle. Notons que l'environnement peut correspondre dans notre modèle de description à l'utilisateur externe et que les actions peuvent être assimilées aux requêtes d'exécution de services. En revanche, les similitudes s'arrêtent là. En effet, dans le cas de l'analyse décisionnelle l'environnement est entièrement libre d'exécuter les actions comme il le souhaite, sans aucune contrainte : il est, en quelque sorte, porteur de "l'intelligence" du système dans sa globalité.

Dans ce contexte et au sens de l'analyse décisionnelle, une automatisation se définit comme une délégation à un système d'automatisation, par l'environnement, de décisions de conduite du processus. Suite à cette délégation, l'environnement devient responsable de décisions qui portent sur des actions nouvelles.

5.2.2. Position du problème d'une automatisation dans le cadre d'une application répartie.

Dans le contexte de l'analyse décisionnelle, une automatisation se définit comme une délégation à un système d'automatisation, par l'environnement, de décisions de conduite du processus. Bien que nous soyons en accord avec cette définition nous la trouvons incomplète. En effet dans le cadre de l'analyse décisionnelle deux problèmes se posent. Le premier est

que l'analyse décisionnelle ne dispose pas d'outils pour décrire lors d'une automatisation quelle finalité précise, parmi toutes les finalités possibles, l'environnement a délégué au système d'automatisation. Le deuxième problème est plus général : il n'est pas possible de savoir, en fonction des contraintes, quelles sont les finalités que peut réaliser un équipement ayant une architecture décisionnelle donnée, il est donc impossible de savoir ce qui peut effectivement être automatisé.

Ainsi dans le cas d'une application répartie construite en interconnectant des instruments intelligents, nous définissons une automatisation comme étant une délégation à un système d'automatisation de décisions de conduite de l'application inscrites dans le cadre d'une finalité précise et autorisée par l'application. Suite à une automatisation, l'utilisateur voit donc apparaître de nouveaux services que nous appelons services composés. Ces derniers représentent les finalités qui ont été automatisées.

Définition 5.1:

On appelle service composé d'une application répartie toute séquence de requêtes d'exécution de service et/ou d'événements permettant à l'utilisateur de suivre un chemin du graphe de menus de l'application considérée (et/ou de rester dans un même menu).

Il ressort donc que le problème de la formulation du procédé d'automatisation d'une application répartie se ramène à la spécification formelle de toutes les combinaisons de commandes autorisées par les contraintes de conduite de l'application. Il s'agit donc de déterminer le langage exprimant toutes les séquences de requêtes d'exécution de services que peut formuler l'utilisateur pour la conduite de l'application. Chacune de ces séquences définit une finalité (service composé) possible qui peut être déléguée à un système d'automatisation.

5.3. Langage de l'utilisateur associé à la conduite d'un ensemble de services sans contraintes.

Nous cherchons à définir le langage dont les mots expriment toutes les commandes que peut formuler un utilisateur pour la conduite d'un ensemble de services sur lesquels il n'existe aucune contrainte de conduite.

5.3.1. Introduction à la théorie des langages et des systèmes formels [BENZ 91]:

On présente, dans cette partie, les notions fondamentales et les formalismes de la théorie des langages auxquels il sera fait appel dans ce chapitre.

5.3.1.1. Système formel

5.3.1.1.1. Définition d'un ensemble

Mathématiquement, il existe principalement deux manières de définir un ensemble 'S'. On peut soit exhiber tous les éléments de cet ensemble S, s'il est fini, soit écrire le formalisme S={x/P(x)} dans lequel x désigne un symbole générique muet et P(x) une propriété. La première façon de définir un ensemble est appelée définition par énumération, la seconde est une définition par compréhension de S.

En informatique théorique, il existe deux grandes manières de définir effectivement un ensemble: la première se fait par un algorithme de reconnaissance qui pour chaque candidat fournit une réponse positive ou négative selon que ce candidat appartient ou non à S, la seconde se fait par un algorithme de production qui calcule des éléments de S. Un objet est alors élément de cet ensemble s'il est produit en un temps fini. Le premier procédé caractérise les ensembles récursifs et fonde la théorie des automates [BENZ 91]. Le second caractérise les ensembles récursivement énumérables et fonde la théorie des grammaires [BENZ 91]. Il repose sur le concept d'induction.

5.3.1.1.2. Ensemble défini par un schéma d'induction

On définit un ensemble par induction, ou par schéma d'induction en trois phases:

- phase initiale ou base : on décide d'un certain nombre d'objets appartenant à S. Ces objets forment un ensemble B appelé base (cet ensemble peut être lui-même défini par un schéma d'induction),
- phase inductive ou règles : il s'agit d'énoncer un certain nombre (fini) de règles de production. Une règle décrit comment à partir de k objets de S on peut former sans ambiguïté un nouvel élément de S,
- phase de clôture : cette phase à pour but de délimiter S.

Exemple :

Soit par exemple l'ensemble S défini par:

phase initiale : sa base est B={a,b}

Phase de production : les règles

R1 : w → wa

R2 : w → bw

Phase de clôture : S= {w/w= b*a*}

(le symbole * indique un nombre quelconque de concaténation de l'entité qu'il indice)

L'ensemble S défini dans l'exemple précédent est dit minimal pour l'inclusion : quel que soit S' dont les éléments sont produits à partir des règles on a S⊇S'. L'ensemble S est dit aussi stable pour les règles en question, en effet l'application de ces règles à des éléments de S produit des éléments de la forme b*a* donc appartenant à S.

5.3.1.2. Langage formel

On présentera dans ce paragraphe les différents opérateurs et les notations employées en théorie des langages.

5.3.1.2.1. Algèbre et combinatoire élémentaire des mots d'un alphabet : la concaténation

Soit V un ensemble que l'on appellera alphabet, cet ensemble est l'ensemble des lettres à partir desquelles on formera des mots du langage. La première opération que l'on peut présenter est la concaténation.

Soient a et b deux éléments de V, on forme alors le mot w comme suit : w=cat(a,b) = ab

Le mot w est alors élément de l'ensemble V^+ défini comme l'ensemble des mots w formés par concaténation des éléments de l'alphabet V. En d'autres termes V^+ est l'ensemble des chaînes de caractères de longueurs $n \geq 1$ dans lesquelles tous les caractères sont des éléments de V. On étend cet ensemble par adjonction de l'élément ε (mot fictif de longueur nulle) à l'ensemble V*. On a: $V^*=V^+\cup\{\varepsilon\}$ et dès lors $n \geq 0$.

La concaténation a comme propriétés:

$\forall w1,w2 \in V^*$ cat(w1,w2)=w1w2 $\in V^*$

$\forall w1 \in V^*$ cat(w1,ε)=w1, cat(ε,w1) =w1

$\forall w1,w2,w3 \in V^*$ cat(cat(w1,w2),w3)=cat(w1, cat(w2,w3))

L'ensemble V* muni de l'opérateur de concaténation est donc muni d'une loi de composition interne. On dit que V* a une structure de monoïde pour la concaténation. Plus précisément V* est appelé monoïde libre engendré par V.

5.3.1.2.2. Langage sur un alphabet : opérations.

Soit V un alphabet quelconque, et soit V* le monoïde libre engendré par V.

Définition 5.2:

On appelle langage formel sur un alphabet V toute partie de V*. On suppose V fini et non vide de sorte que V* soit dénombrable (en bijection avec l'ensemble des entiers).

5.3.1.2.2.1. Le produit de langages

Soient deux langages L et L' sur V.

On appelle produit de ces deux langages le langage L'' noté :

$L''=LL'=\{\alpha / \alpha=\beta\beta' \ \beta \in L \ \beta' \in L'\}$.

5.3.1.2.2.2. La somme de deux langages

On définit également la somme de deux langages L'' : $L''= L+L'=\{\alpha / \alpha \in L$ ou $\alpha \in L'\}$.

On définit pour tout langage L le langage L^k de la manière suivante :

$L^0=\varepsilon$; $L^k=L^{k-1}.L$ On montre alors que $L^* = \bigcup_{k\geq 0} L^k$

(L* représente le langage des concaténations quelconques des mots de L)

5.3.1.2.2.3. Propriétés équationnelles

Pour des raisons de commodité et parce que l'on est familiarisé avec ce type de formalisme on notera le produit par l'opérateur '.' et l'union par l'opérateur '+'. Il est facile de voir que ces opérateurs ont des propriétés similaires à la multiplication et à l'addition dans l'ensemble des nombres réels. Ainsi, le formalisme introduit ici offre des facilités de manipulation des expressions. Une propriété équationnelle remarquable de ces opérateurs est donnée par le théorème suivant.

Théorème 5.1 [BENZ 91]:

Soient deux langages A et B sur un vocabulaire V, alors le langage B.A* est solution de l'équation (à inconnue le langage Z sur V): Z=Z.A+B.

Quand $\varepsilon \neq A$ cette solution est unique.

5.3.1.2.3. Langage rationnel et automate fini

Un langage L engendré sur un vocabulaire V par les opérateurs de produit, d'itération et d'union et dont les mots de L peuvent se noter sous la forme d'une expression régulière est un langage dit rationnel. Un exemple d 'expression régulière est : a(ba*+c)ac*

L'ensemble des langages rationnels forme la classe RAT des langages rationnels.

Définition 5.2

On appelle automate fini (en général non-déterministe) sur un alphabet ou un vocabulaire V, la donnée d'un quintuplet G(V,q,i,t,f) où q est l'ensemble fini des états de l'automate, $i \subseteq q$ ($t \subseteq q$) est le sous-ensemble des états initiaux (terminaux), $F \subseteq q \times V \times q$ est le sous-ensemble des transitions de l'automate.

Selon les notations du chapitre précédent G=[q,F] est le graphe états transitions correspondant

Le langage de l'automate G est noté L(G). Ce langage est constitué des mots w tels que les caractères qui les forment sont ceux des étiquettes des arcs de l'automate rencontrées lors d'un parcours de ce dernier et ce dans le même ordre.

Théorème 5.2 [BENZ 91] :

Soit L un langage sur V, les propositions suivantes sont équivalentes:

1) L est reconnu par un automate fini déterministe

2) L est reconnu par un automate fini non-déterministe

3) L est rationnel.

On montre que pour un langage rationnel L il existe un automate fini déterministe qui reconnaît ce langage. La réciproque est vraie. Cet automate est appelé automate équivalent. (l'équivalence entre une machine à état et les différents types de grammaires est démontrée dans [BENZ 91])

5.3.2. Langage des requêtes de l'utilisateur relatif à un ensemble de services.

Rappelons que notre objectif est de définir le langage dont les mots expriment toutes les commandes que peut formuler un utilisateur pour la conduite d'un ensemble de services sur lesquels il n'existe aucune contrainte de conduite.

Les ordres que peut formuler l'utilisateur se résument à des séquences d'émission de requêtes plus ou moins espacées dans le temps, ces ordres sont donc des mots formés par des alternances de concaténations de mots des deux langages suivants :

Le premier exprime les requêtes que peut formuler l'utilisateur .

Le second exprime le temps qui peut s'écouler entre les émissions de deux requêtes consécutives.

Soit S_G l'ensemble de tous les services d'un graphe des menus.

Soit Ri la requête relative au service si $\in S_G$:

Soit V_R le vocabulaire formé par tous les Ri

V_R = {Ri , / Ri requête sur si $\in S_G$} et V_R* l'ensemble des mots formés par des concaténations d'éléments de V_R.

Et soit T le langage de représentation décimale de la variable temps (exemple :10s).

Définition 5.3:

On appelle langage de l'utilisateur $Lu(S_G)$ associé à l'ensemble de service S_G toute concaténation quelconque de mots de V_R* ou de T.

Remarques

1 un tel langage peut être défini pour n'importe quel ensemble de services en particulier pour chaque menu d'un graphe

2 exemple de mot : R1/5s/R2/10s/R3

Proposition 5.1:

$Lu(S_G) = (V_R^* + T)^*$

Démonstration

Un mot de $Lu(S_G)$ est une concaténation quelconque de mots de V_R^* ou de T

$Lu(S_G)$ vérifie donc l'équation :

$Lu(S_G) = Lu(S_G).V_R^* + Lu(S_G).T+\varepsilon= Lu(S_G).(V_R^* + T) +\varepsilon$

la solution est donc :

$Lu(S_G) = (V_R^* +T)^*.\varepsilon =(V_R^* + T)^*$

ce résultat est prévisible en fait $(V_R^* + T)^*$ est le langage des concaténations quelconques de mots de V_R^* ou de T

Remarque :

On confondra par abus de notation le mot w de $Lu(S_G)$ avec le fait que l'utilisateur émet les requêtes exprimées par ce mot.

5.4. Sous graphe des menus contrôlé par l'utilisateur

On considère maintenant la conduite d'un ensemble de services sur lesquels existent des contraintes relatives à l'application. Dans ces conditions, les services sont structurés suivant un graphe des menus qui représente la traduction de ces contraintes.

5.4.1. Evénements d'un graphe de menu dépendant de l'utilisateur:

Parmi les événements conditionnant les différentes transitions d'un graphe de menus nous distinguons ceux dont l'occurrence ne dépend que des choix faits par l'utilisateur. Le franchissement des transitions correspondantes peut donc être planifié à l'avance. Ces différentes transitions peuvent définir des chemins dont le parcours est automatisable.

Définition 5.4:

Soit G(X,U) un graphe de menus et soit e_{ij} un événement tel que $(i, e_{ij}, j) \in U$.

e_{ij} est dit dépendant de l'utilisateur si et seulement si l'assertion qui exprime l'événement n'est fonction que du langage de commande de l'utilisateur c'est-à-dire fonction de mots w tels que $w \in Lu(S_G)$.

5.4.2. Sous graphe du graphe des menus dépendant de l'utilisateur:

Pour pouvoir planifier à l'avance le cheminement dans le graphe des menus, nous transformons ce dernier en un graphe ne concernant que des transitions dont les événements de franchissement ne dépendent que de l'utilisateur.

Définition 5.5:

Soit G =[X,U] un graphe de menus, nous appelons sous graphe des contraintes dépendant de l'utilisateur G(u)=[X,U_u] avec U_u={T=(i,e_{ij},j)∈U /e_{ij} dépend de l'utilisateur}.

Soit A(u) la matrice d'incidence de ce nouveau graphe.

5.5. Langage des séquences automatisables dans le graphe des menus d'une application.

5.5.1. Rappel sur l'algèbre des chemins dans un graphe :

Dans cette partie nous introduisons, tout en les appliquant directement à notre problème, les notions principales de l'algèbre des chemins [GOND 79]. Cette algèbre représente l'ensembles des structures mathématiques appropriées pour résoudre les problèmes de cheminement dans un graphe. Dans cette algèbre, la détermination de l'expression des mots du langage représentant les chemins possibles dans un graphe, se ramène à la résolution d'une équation linéaire du premier ordre, cette résolution est faite dans une classe de chemins: les dioïdes.

5.5.1.1. Définition et propriétés :

5.5.1.1.1. Une classe d'algèbre des chemins : les dioïdes

On considère le dioïde (V* , + , .) c'est-à-dire l'ensemble V* des mots sur un alphabet V muni des deux lois '+' et '.' tel que:

- L'opération '+' est appelée « addition » et représente dans notre cas l'union ensembliste, elle munit V* d'une structure de monoïde et admet comme élément nul 0 = ∅.
- L'opération '.' est dite multiplication, dans notre cas elle correspond à la concaténation, elle admet comme élément neutre ε, qui représente la chaîne vide

- La multiplication est distributive à droite et à gauche par rapport à l'addition et admet l'élément nul comme élément absorbant.

On définira alors l'addition et la multiplication pour l'ensemble $M_n(V^*)$ des matrices carrées d'ordre n à éléments dans V* à partir des lois '+' et '.' de la même façon que pour les matrices à éléments réels :

$$A+B = (a_{ij}+b_{ij})$$

et $A.B = (\sum_k a_{ik}.b_{kj})$ où $\sum_k$ désigne la sommation sur 'k' et par rapport à '+'

On vérifie facilement que l'ensemble des matrices $M_n(V^*)$ muni de ces deux lois est un dioïde avec:

comme élément nul la matrice : $\Sigma = (\Sigma_{ij} = 0)$

et comme matrice unité la matrice : $E=(e_{ij} = \varepsilon$ si i=j, 0 autrement) on notera que $A^0(G)=E$

5.5.1.1.2. Matrice d'incidence d'un graphe de menus :

Considérons un graphe de menus G[X,U] et V l'alphabet formé par les événements des différentes transitions de U. On peut définir pour ce graphe une matrice d'incidence A(G) à éléments dans V et représentant les événements de passage d'un menu à l'autre :

Définition 5.6:

On appelle matrice d'incidence d'un graphe de menus G=[X,U]la matrice A(G) telle que :

$A(G) = (a_{ij})$

avec $a_{ij}=e_{ij}$ si $(i,e_{ij},j) \in U$; $a_{ij}=0$ autrement.

On peut aussi définir un chemin de G :

Définition 5.7:

On appelle chemin une suite de sommets de X se succédant dans le graphe. Ainsi à tout chemin $u=(i_1,i_2, ...,i_k)$ on associe le mot $w(u)= e_{1,2}, e_{2,3}, ..., e_{k-1,k}$ qui est la suite des événements correspondant aux étiquettes des transitions rencontrées lors du parcours de u.

Il est facile de voir que l'ensemble des mots w(u) correspond au langage du graphe G [Déf.5.3] c'est-à-dire le langage constitué par des mot de V* et dont les caractères correspondent à ceux des événements des transitions du graphe rencontrées lors d'un parcours de ce dernier et ce dans le même ordre.

on notera :

C_{ij}^{k} : l'ensemble des chemins de i à j formés d'une suite de k+1 sommets pas forcément distincts

$C_{ij}^{(k)}$: l'ensemble des chemins de i à j formés d'une suite d'au plus k+1 sommets pas forcément distincts

Un grand nombre de problèmes de cheminement se ramènent au calcul de :

$A^k(G)$

ou de : $A^{(k)} = E+A+A^2+...+A^k$

Les éléments des deux dernières matrices correspondent à des chemins particuliers, en effet la proposition suivante a été démontrée dans [GOND 79]:

Proposition 5.2:

$A_{ij}^{k} = \sum_{u \in C_{ij}^{k}} w(u)$

$A_{ij}^{(k)} = \sum_{u \in C_{ij}^{(k)}} w(u)$

où A_{ij}^{k} et $A_{ij}^{(k)}$ dénotent les termes ij des matrices A^k et $A^{(k)}$ respectivement

5.5.1.2. Quelques théorèmes de convergence

Parmi les travaux de Gondrand sur l'algèbre des chemins, nous nous intéressons à la résolution d'équations du premier degré. Dans cette partie, nous allons rappeler ces résultats et introduire les notations nécessaires.

Pour chaque mot $w \in V^*$ on définit : $w^{(k)} = \varepsilon + w^1 + w^2 + ... + w^k$

w est dit p-régulier si $w^{(p)} = w^{(p+1)}$ on a alors dans ce cas : $w^{(p+1)}=w^{(p+2)} ...=w^{(p+q)} \ \forall \ q \geq 1$

Pour chaque élément p-régulier de V^* on déduit l'existence de w^* le quasi inverse de w défini par $w^* = \lim_{k \to \infty} w^{(k)}$

Il est à noter que cette notion de régularité de mot de V^* est relative à un langage (en l'occurrence celui d'un graphe) ; ce sont les règles du langage (en l'occurrence les transitions du graphe) qui déterminent si un mot est p-régulier ou non. Notons que si un mot w est p-régulier alors tous les mots formés d'une concaténation de plus de p fois le mot w sont

identiques à w. Si w est 0-régulier alors w* représente tous les mots formés d'un nombre quelconque de concaténations du mot w.

On définit un circuit dans un graphe G comme étant un chemin dont les sommets départ et arrivée sont identiques. On appellera circuit pointé la donnée d'un circuit et de son point de départ.

Définition 5.8:

On dira qu'un graphe G est sans circuit p-absorbant si le mot associé à chacun des circuits pointés du graphe est p-régulier.

Cette notion de p-absorbant reflète le fait que la sémantique donnée au franchissement des arcs est régulière à partir d'un certain ordre (nombre de franchissement de transitions). Il est facile de voir que lorsqu'il s'agit de déterminer l'expression des chemins dans un graphe ce dernier est 0-absorbant (c'est-à-dire que le fait de boucler sur un circuit fermé quelconque du graphe à toujours la même interprétation quel que soit le nombre de boucles effectuées).

Théorème de Gondrand :

Si G=[X,U] est un graphe p-absorbant et si '.' est commutative alors $A^{(k)}$ admet une limite A* quand k tend vers l'infini $A^* = \lim A^{(k)}$ vérifie l'équation :

(1) $A^*=AA^* +E$

En multipliant à gauche (respectivement à droite) l'équation (1) des deux cotés par une matrice B et en posant Y=A*.B (respectivement Z=B.A*) on a :

(2) $Y=A.Y+B;$

(3) $Z=Z.A+B$

Il apparaît donc que A*B (respectivement BA*) est une solution de (2) (respectivement 3) il est en plus démontré dans [GOND 79 pp 72-73] que cette solution est minimale c'est-à-dire que quel que soit Y solution il existe C tel que Y=C+A*.B (Z=C+B.A*)

La matrice A* est la matrices des expressions générales des chemins permettant de passer d'un sommet à l'autre dans le graphe.

Par la suite, nous aurons besoin de l'expression des mots permettant de quitter un sommet quelconque.

Définition 5.9:

Soit G un graphe p-absorbant, nous appelons $L_S(G)$ le vecteur colonne (de dimension le nombre de sommets) formé des langages tels que les mots de chacun représentent les chemins permettant de quitter un sommet du graphe G.

$$L_S(G) = \begin{vmatrix} L_{s1}(G) \\ L_{s2}(G) \\ \dots \\ \\ L_{sn}(G) \end{vmatrix}$$

Nous appelons $L_S(G)$ le vecteur des chemins sortants.

Il est possible d'exprimer les composantes du vecteur des chemins sortants, en effet ce dernier est solution d'une équation du même type que (2).

Proposition 5.3:

Soit G un graphe p-absorbant, et A la matrice d'incidence de G. $L_S(G)$ vérifie alors l'équation : $L_S(G) = A(G).L_S(G) + B$ avec :

$$B = \begin{vmatrix} \varepsilon \\ \varepsilon \\ \dots \\ \\ \varepsilon \end{vmatrix}$$

d'où $L_S(G) = A^*(G).B$ est une solution.

Ainsi le langage L(G) du graphe G [§5.3.1.2.3]vérifie :

$L(G) = B^T.\ L_S(G) = B^T.A^*(G).B$

Démonstration

Le langage des chemins sortants d'un sommet "i" peut être vu comme étant la somme de toutes les concaténations entre un événement eij et le langage des chemins sortants du sommet "j".

Ainsi, les composantes du vecteur LS(G) vérifient :

$Ls1(G) = e11\ Ls1 + e12\ Ls2 + \ + e1n\ Lsn + \varepsilon$

$\vdots$

$Lsn(G) = en1\ Ls1 + en2\ Ls2 + \ + enn\ Lsn + \varepsilon$

Nous avons donc $LS(G) = A(G).LS(G) + B$.

D'où d'après Gondrand LS(G) =A*(G).B est une solution.

Notons que l'expression générale d'un chemin quelconque de G correspond à un mot de l'une des composantes du vecteur LS(G) ainsi le langage de G tel que défini par la théorie des langages [§5.3.1.2.3] est la somme des éléments de LS(G) d'où :

L(G) = BT.A*(G).B avec BT le vecteur transposé de B.

Du moment où A* a été déterminée [GONDO 79], ce dernier résultat est prévisible. En effet un mot quelconque de L(G) est l'un des élément de A*(G) et l'expression la plus générale d'un mot de L(G) est donc la sommation de tous les éléments de A*(G).

5.5.2. Langage des séquences contrôlables par l'utilisateur

Définition 5.10:

On dit qu'un chemin dans un graphe de menus, c'est-à-dire une séquence d'événements, est automatisable si et seulement si tous les événements dépendent de l'utilisateur.

Nous cherchons le langage des mots exprimant des chemins automatisables dans le graphe des menus G. Ce langage représente des chemins dont le parcours peut être planifié à l'avance. Il est facile de voir que ce langage est celui des chemins du sous-graphe des menus dépendant de l'utilisateur.

En appliquant le théorème de Gondrand avec les notations de [§5.4] ce langage vérifie $LG(u)=B^T.A^*(u).B$

Nous rappelons que $L_S(G(u))$ est le vecteur des chemins sortants de G(u) ce vecteur sera utile par la suite.

5.5.3. Langage de l'utilisateur propre à un menu.

Soit G=[X,U] un graphe des menus. Nous cherchons le langage L_μ des séquences de requêtes que peut formuler l'utilisateur dans un menu μ sans pour autant quitter ce dernier.

D'après [§5.3.2 ; proposition 5.1] à chaque menu on peut associer un langage Lu(μ) de toutes les séquences que peut formuler l'utilisateur dans ce menu.

Soit $e(\mu)=\{e/\ T(\ ,e,\ )\ \in\ U\ \}$ l'ensemble des événements conditionnant des transitions sortantes du menu μ dans le graphe des menus

Le langage L_μ est l'ensemble des mots que peut formuler l'utilisateur sans qu'un événement de $e(\mu)$ ne se produise. L_μ est donc l'ensemble des mots dont ne dépend aucun événement de $e(\mu)$, d'où la définition de L_μ :

Définition 5.11:

On appelle langage propre à l'utilisateur d'un menu

$L_\mu = \{w \in Lu(\mu) / \forall e \in e(\mu), e \text{ est indépendant de } w\}$

Si l'on considère le langage propre associé à chacun des menus, ont peut définir le vecteur :

$$L_\mu(G) = \begin{vmatrix} L_{\mu 1} \\ L_{\mu 2} \\ \dots \\ L_{\mu n} \end{vmatrix}$$

5.5.4. Services composés ou services automatisables d'un graphe de contraintes.

Définition 5.12:

Soit G un graphe de menus d'une application et soit G(u) le sous graphe dépendant de l'utilisateur. On appelle service composé toute séquence de requêtes et/ou d'événements dépendant de l'utilisateur et permettant de suivre un chemin dans le graphe G(u).

Proposition 5.4:

Soit G le graphe des menus d'une application et soit G(u) le sous graphe dépendant de l'utilisateur. Soient aussi Ls(G(u)) le vecteur des chemins sortants de G(u) et $L_\mu(G(u))$ le vecteur des langages propres à chaque menu de G(u).

Le langage exprimant les services composés de l'application considérée est :

$L(\text{services composés}) = Ls(G(u))^T . L_\mu(G(u))$

Démonstration :

Pour chaque menu (μi) les expression possibles d'un service composé permettant un cheminement à partir de ce menu sont toutes les concaténations de mots de $L_{\mu i}$ permettant de ne pas quitter le menu suivies d'un mots de $L_{S\mu i}$ permettant de quitter ce menu. Ainsi le langage des mots exprimant un service composé permettant un cheminement à partir d'un menu (μi) est le produit : $L_{\mu i}\, L_{S\mu i}$.

La sommation à tous les menus des ($L_{\mu i}$ $L_{S\mu i}$) définit donc le langage de l'ensemble des services composés possibles, d'où :

L(services composés) $=\Sigma_i$ $L_{\mu i}$ $L_{S\mu i}$ $=Ls(G(u))^T.L_\mu(G(u))$.

5.6. Conclusion

Dans ce chapitre, nous avons étudié la possibilité de déléguer des séquences de conduite d'une application à un système d'automatisation. Notre objectif était de définir des niveaux de commande supérieurs permettant à l'utilisateur une conduite plus élaborée, par le biais de services composés, représentant des séquences d'exécution de services de base des différents instruments d'une application donnée.

Pour résoudre le problème, notre démarche part de la constatation que les éventuels services composés pouvant être définis correspondent à des chemins du graphe des menus dont le parcours peut être planifié à l'avance. Ainsi, nous avons cherché le langage dont les mots expriment des séquences d'exécution de services que peut formuler l'utilisateur et dont le cheminement dans le graphe des menus de l'application ne dépend que d'événements contrôlables.

Nous avons ainsi obtenu l'expression de tous les services composés pouvant être définis donc l'expression des séquences de conduite pouvant être déléguées à un système d'automatisation.

Chapitre 6. Exemple

6.1. Introduction

Dans les chapitres précédents, nous avons démontré l'existence d'un système de transitions permettant d'exprimer pour un utilisateur les contraintes relatives à la conduite d'une application construite en interconnectant un ensemble d'instruments intelligents. Ce système est un graphe états transitions que nous appelons graphe des menus, il permet de spécifier la façon dont le choix de l'utilisateur est limité par la finalité de l'application. Ce graphe permet d'obtenir à toutes les étapes de fonctionnement la liste des services activables par l'utilisateur. Une fois construit, le graphe des menus permet aussi de trouver les séquences dont l'exécution peut être planifiée à l'avance donc déléguée à un système d'automatisation.

Le but de cette partie est d'illustrer par un exemple la façon dont les résultats obtenus s'appliquent à un cas concret. Afin de pouvoir faire une comparaison avec d'autres méthodes de modélisation on a choisi un exemple de la littérature. Il s'agit d'un exemple traité dans plusieurs articles différents [BUCK 94] [BUCK 91 a] [BUCK 91 b] [BUCK 91 c] et utilisé par ses auteurs pour illustrer la démarche d'automatisation par analyse décisionnelle.

Dans ce chapitre, on se propose, tout d'abord, d'exprimer les contraintes relatives à l'installation de manutention choisie comme exemple, c'est-à-dire de retrouver le graphe des menus permettant de définir les utilisations cohérentes de l'ensemble de l'installation. Ensuite, nous suivrons le plan des auteurs de l'exemple dans leur démarche d'automatisation du processus. Notre objectif est de retrouver formellement les séquences effectivement automatisables afin de remplacer la démarche des auteurs, qui relève d'avantage de l'expérience, par une démarche formelle elle-même automatisable.

6.2. Préambule.

L'exemple choisi a été utilisé par ses auteurs [BUCK 94] pour illustrer la démarche d'automatisation par analyse décisionnelle. Rappelons que celle ci s'inscrit dans la lignée des conceptions orientées objets, elle qualifie un équipement comme un ensemble d'objets immergés dans un environnement de processus [BUCK 94]. Dans ce sens, l'environnement est chargé de diriger et de contrôler ce processus par rapport à une finalité dont il est porteur et ceci par les actions qu'il entreprend. Vu du processus, l'environnement n'a aucune forme matérielle. En fait l'environnement correspond dans notre modèle à l'utilisateur externe, à la différence que dans le cas de l'analyse décisionnelle ce dernier est libre d'exécuter les actions comme il le souhaite sans aucune contrainte.

Les auteurs de l'exemple rappellent qu'une analyse des modes de marche s'impose pour assurer un fonctionnement cohérent et optimal dans toutes les situations pouvant se produire. Le problème est que ces auteurs ne définissent ni ce que sont les situations qui peuvent se produire ni ce qu'ils appellent un fonctionnement cohérent. En effet bien que la démarche d'automatisation par analyse décisionnelle n'exclue pas l'existence de procédures de fonctionnement qui doivent être respectées et qui peuvent imposer un ordre dans les exécutions des actions, cette démarche d'automatisation n'a prévu aucun outil pour décrire cette situation. C'est à l'environnement qu'incombe la tâche de ne pas se tromper et de reconnaître les utilisations cohérentes : l'environnement est en quelque sorte porteur de "l'intelligence" du système dans sa globalité.

Toutefois, exprimer les contraintes n'est peut-être pas l'objectif de l'analyse décisionnelle cependant celle-ci définit l'automatisation comme étant la délégation à un système d'automatisation par l'environnement de décisions concernant la conduite. Cette définition de l'automatisation serait parfaite s'il ne fallait pas distinguer les séquences de prise de décision effectivement réalisables par l'équipement de celles qui ne le sont pas.

Ainsi, ce qui fait défaut dans la démarche d'automatisation par analyse décisionnelle est l'absence de relation entre la finalité portée par l'environnement et les possibilités réelles de l'équipement : il n'existe pas d'outil pour exprimer les contraintes de conduite imposées par une installation donnée. Dés lors la finalité de l'environnement pourrait causer la destruction de l'équipement, ou l'équipement pourrait être, tout simplement, incapable de satisfaire certaines finalités.

6.3. Description de l'exemple

6.3.1. Le processus

Une installation de manutention comporte deux trémies contenant chacune un produit diffèrent. Chaque produit se déverse au moyen d'un extracteur E1 ou E2 sur un tapis A préalablement mis en rotation et, de là, sur un tapis B pouvant tourner dans les deux sens de façon à déverser le produit extrait dans le silo S1 ou dans le silo S2, figure 6.1.

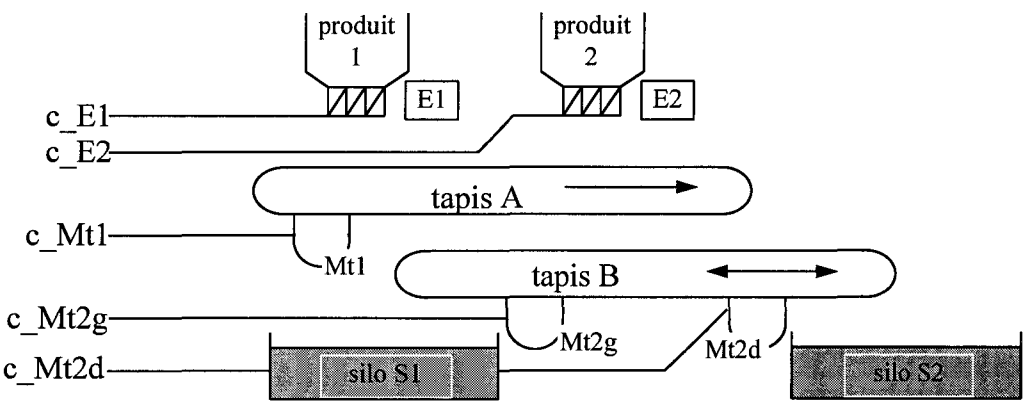


Figure 6.1 : Exemple

6.3.2. Les contraintes de conduite (au niveau de l'environnement)

De façon à limiter les pointes d'appel de courant et tenir compte des temps respectifs de mise en route, la séquence de démarrage doit s'effectuer dans l'ordre suivant :

- Rotation du tapis B dans le sens choisi : G-vers silo S1 ; D vers silo S2
- 5 secondes après, mise en rotation du tapis A
- 10 secondes après : mise en marche du (ou des) extracteur(s) E1 et E2

L'arrêt normal s'effectue par la procédure inverse.

- Arrêt immédiat des extracteurs E1 et E2
- 10 secondes après arrêt du tapis A
- 5 secondes après , arrêt du tapis B.

En fait les tapis n'ont aucun pouvoir de stockage, il faut donc éviter de déverser du produit si l'un des tapis est à l'arrêt.

6.3.3. Commande de l'équipement : l'ensemble des services.

L'environnement extérieur agit sur l'installation à l'aide des commutateurs :

actionneur 1	c_E1	:	Marche / Arrêt de l'extracteur E1
actionneur 2	c_E2	:	Marche / Arrêt de l'extracteur E2
actionneur 3	c_Mt1	:	Marche / Arrêt du tapis A
actionneur 4	c_Mt2g	:	Marche Gauche / Arrêt du tapis B
	c_Mt2d	:	Marche Droite/ Arrêt du tapis B

Les décisions concernant la conduite du processus appartiennent toutes à l'environnement, elles se définissent comme suit :

D0 : Démarrage de l'extracteur E1 ; D1 : Arrêt de l'extracteur E1
D2 : Démarrage de l'extracteur E2 ; D3 : Arrêt de l'extracteur E2
D4 : Mise en rotation du tapis A ; D5 : Arrêt du tapis A
D6 : Mise en rotation gauche du tapis B ; D7 : Arrêt de la rotation gauche du tapis B
D8 : Mise en rotation droite du tapis B ; D9 : Arrêt de la rotation droite du tapis B

6.3.4. Les ressources

Les ressources sont les entités matérielles et informationnelles utilisées par l'équipement, dans notre cas on dispose de :

Un tapis A ; Un tapis B ; Un extracteur E1 ; Un extracteur E2 ; Un silo S1 ; Un silo S2

Des commutateurs :

c_E1 ; c_E2 ; c_Mt1 ; c_Mt2g ; c_Mt2d

L'ensemble des ressources de notre exemple est donc :

R = { Tapis A ; Tapis B ; Extracteur E1; Extracteur E2 ; Silo S1 ; Silo S2 ; Commutateurs: c_E1 ; c_E2 ; c_Mt1 ; c_Mt2g ; c_Mt2d }.

6.3.5. Graphe des modes d'utilisation propre à chaque équipement du processus

Dans le chapitre 2, nous avons défini un mode d'utilisation comme étant un ensemble de services cohérents. Dans le cas de notre installation de manutention, la conduite des services des différents instruments (actionneurs) doit être organisée en graphes de modes d'utilisation. Pris séparément, chacun des équipements de l'installation de manutention a son propre graphe des modes d'utilisation. Intuitivement, on se rend compte que chacun des équipements (actionneurs) présente principalement deux modes d'utilisation :

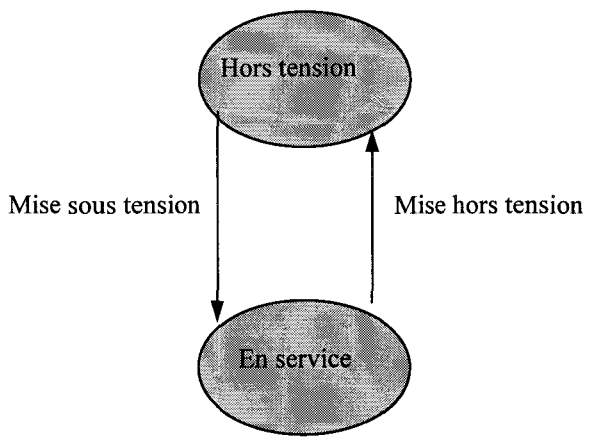


Figure 6.2: modes d'utilisation des équipements

Les services mise sous tension et mise hors tension sont des services de changement de mode. Par ailleurs, les significations des modes d'utilisations sont les suivantes :

Extracteur E1 : Hors tension ={mise sous tension}

En service ={D0, D1, mise hors tension}

Extracteur E2 : Hors tension ={mise sous tension}

En service ={D2, D3, mise hors tension}

Tapis A : Hors tension ={mise sous tension}

En service ={D4, D5, mise hors tension}

Tapis B : Hors tension ={mise sous tension}

En service ={D6, D7, D8, D9, mise hors tension}

Remarques

1 Si les mises hors tension et mises en service sont pilotées par des interrupteurs différents, il faut s'assurer que tous les modes d'utilisation sont "En service" pour l'instrument composé global. Ceci se fait automatiquement si tout est piloté par un instrument unique.

2 On se place par la suite dans le mode d'utilisation global où tous les actionneurs sont "En service".

6.4. Expression des contraintes de conduite dans un graphe de menus d'utilisation.

6.4.1. Position du problème

La description des contraintes faite par les auteurs est typique de ce qui se fait dans la pratique, à savoir une rédaction textuelle que l'environnement est supposé lire, comprendre, et appliquer. Ce type de description n'admet pas de formulation générale permettant à un environnement quelconque (humain ou non) d'identifier de façon précise les fonctionnalités qu'offre un équipement donné, il est ainsi difficile d'utiliser un équipement avec lequel on n'est pas familiarisé. Ce type de description peut même poser des problèmes d'interprétation.

Le problème qui se pose est d'exprimer les contraintes décrites par les auteurs en organisant les services proposés par l'installation de manutention dans un graphe de menus de façon à tenir compte des procédures de démarrage et d'arrêt imposées par l'équipement. Ce graphe permet à une entité extérieure de pouvoir utiliser l'équipement dans le cadre de la finalité qu'elle souhaite tout en respectant les contraintes imposées par l'équipement.

6.4.2. Les services proposés par l'équipement à une entité extérieure.

Les décisions concernant la conduite du processus appartiennent toutes à l'environnement [BUCK 94]. En fait l'ensemble des décisions que peut prendre l'environnement au niveau de la commande directe des actionneurs correspond pour notre modèle de description à l'ensemble des services que propose l'équipement à une entité extérieure, ainsi l'ensemble de ces services est :

S = { D0 ; D1 ; D2 ; D3 ; D4 ; D5 ; D6 ; D7 ; D8 ; D9 }

avec :

D0 : Démarrage de l'extracteur E1 ; D1 : Arrêt de l'extracteur E1

D2 : Démarrage de l'extracteur E2 ; D3 : Arrêt de l'extracteur E2

D4 : Mise en rotation du tapis A ; D5 : Arrêt du tapis A

D6 : Mise en rotation gauche du tapis B ; D7 : Arrêt de la rotation gauche du tapis B

D8 : Mise en rotation droite du tapis B ; D9 : Arrêt de la rotation droite du tapis B

Remarque :

> Dans le but de garder la même notation que les auteurs de l'exemple nous noterons Di le service qui correspond dans le cas de l'analyse décisionnelle à la décision Di que peut prendre l'environnement.

6.4.3. Expression des contraintes : notion de menus d'utilisation.

Dans le chapitre 4, on exprime les contraintes par des événements conditionnant les possibilités de proposer les services à l'utilisateur. L'existence de ces événements permet de démontrer qu'on peut exprimer les contraintes par un graphe de menus, un menu étant un ensemble de services activables. Dans un menu l'utilisateur peut exécuter le service de son choix sans aucune contrainte. Le passage d'un menu à l'autre est conditionné par les événements susceptibles de modifier l'ensemble des services activables par l'utilisateur.

Nous ne proposons pas de méthode pour retrouver les événements, ces derniers peuvent être de toutes natures ce qui rend très difficile leur description dans un cadre général. Néanmoins afin de voir à quoi ils peuvent correspondre, on se propose de les retrouver dans le cadre de l'installation de manutention prise comme exemple.

Les contraintes de conduite décrites par les auteurs reflètent le fait qu'il faut limiter les pointes d'appel de courant. Si on étudie bien les procédures de démarrage et d'arrêt de l'installation on se rend compte que les tapis n'ont en fait aucun pouvoir de stockage c'est pourquoi on ne déverse jamais un des produits sur un tapis à l'arrêt et on s'assure que les tapis sont bien vides avant et après usage. Les contraintes ainsi décrites se traduisent par des événements conditionnant les décisions que peut prendre l'utilisateur, l'ensemble de ces événements définit l'ensemble des contraintes de l'équipement.

Le tableau suivant, est un recensement de l'ensemble des événements conditionnant les différents services proposés par l'installation à un utilisateur externe. Ce tableau représente

sur une même ligne : l'événement, sa description et les services qu'il conditionne, la troisième colonne donne la valeur de l'activabilité [ch 4 §4.3.2] affectée à chacun de ces services ceci à chaque fois que l'événement se produit.

Evénement	Description	Services affectés par l'occurrence de l'événement
Ev-1	mise en rotation droite du tapis B c.à.d une exécution de D8	A (D8) = 0 A (D9) = 1
Ev-2	mise en rotation gauche du tapis B c.à.d une exécution de D6	A (D6) = 0 A (D7) = 1
Ev-3	arrêt rotation droite du tapis B c.à.d une exécution de D7	A (D7) = 0 A (D6) = 1 A (D8) = 1
Ev-4	arrêt rotation gauche du tapis B c.à.d une exécution de D9	A (D9) = 0 A (D6) = 1 A (D8) = 1
Ev-5	rotation droite du tapis B depuis 5 secondes c.à.d 5s après une exécution de D8	A (D4) = 1
Ev-6	rotation gauche du tapis B depuis 5 secondes c.à.d 5s après une exécution de D6	A (D4) = 1
Ev-7	rotation droite du tapis B et rotation du tapis A c.à.d une exécution de D4 après E3	A (D5) = 1 A (D9) = 0
Ev-8	rotation gauche du tapis B et rotation du tapis A c.à.d une exécution de D4 après E4	A (D5) = 1 A (D9) = 0
Ev-9	E7 et arrêt du tapis A c.à.d une exécution D5 est l'événement qui suit l'occurrence de E7	A (D5) = 0 A (D4) = 1 A (D9) = 1
Ev-10	E8 et arrêt du tapis A c.à.d une exécution D5 est l'événement qui suit l'occurrence de E8	A (D5) = 0 A (D4) = 1 A (D7) = 1
Ev-11	10 secondes après E5	A (D0) = 1 A (D2) = 1
Ev-12	10 secondes après E6	A (D0) = 1 A (D2) = 1
Ev-13	une exécution de D0	A (D0) = 0 A (D1) = 1
Ev-14	une exécution de D2	A (D2) = 0 A (D3) = 1
Ev-15	A(D0) = A(D2) = 1 depuis plus de 10 secondes	A (D5) = 1
Ev-16	une exécution de D5 quand A(D0) = A(D2) = A(D5) = 1	A (D0) = 0 A (D2) = 0 A (D5) = 0 A (D4) = 1
Ev-17	une exécution de D4 dans les 5 secondes qui suivent l'occurrence de E16	A (D0) = 1 A (D2) = 1 A (D5) = 1 A (D4) = 0
Ev-18	Tapis B en rotation droite (D8 a été exécuté) et 5 secondes se sont écoulées depuis E16	A (D9) = 1
Ev-19	Tapis B en rotation gauche (D6 a été exécuté) et 5 secondes se sont écoulées depuis E16	A (D7) = 1

6.4.4. Construction du graphe des menus

La méthode utilisée est celle décrite dans le chapitre [§4.3.9]. C'est une méthode récurrente se basant sur une procédure permettant de déterminer à un moment donné l'ensemble des services activables et celui des événements qui peuvent se produire. Chacun des événements ainsi déterminés marque par son occurrence le début d'une nouvelle étape. Pour chacune de ces étapes il faut relancer la procédure. Comme pour toute procédure récurrente il faut garantir son arrêt. Dans notre cas, le fait que le nombre des étapes possibles est fini garantit l'arrêt de la procédure et l'obtention du graphe des menus.

Etape 1 de fonctionnement

a/ Recensement de l'ensemble des services activables

Conformément à la procédure de démarrage, à ce stade du fonctionnement l'utilisateur peut uniquement choisir un sens de rotation pour le tapis B. Les services disponibles à cette étape sont donc D6 et D8. Le menu μ_1 est donc constitué des services D8 et D6.

	D0	D1	D2	D3	D4	D5	D6	D7	D8	D9
Services disponibles	0	0	0	0	0	0	1	0	1	0

b/ Recensement des événements du premier niveau

Les événements susceptibles de modifier l'ensemble des services disponibles (c-à-d de modifier le comportement) sont :

Ev-1 : Une exécution de D8. Celle-ci nous conduit à une nouvelle étape : étape 2

Ev-2 : Une exécution de D6. Celle-ci nous conduit à une nouvelle étape : étape 3

Etape 2 de fonctionnement

Cette étape du fonctionnement n'a pas encore été décrite, il faut relancer la procédure.

a/ Recensement de l'ensemble des services disponibles.

	D0	D1	D2	D3	D4	D5	D6	D7	D8	D9
Services disponibles	0	0	0	0	0	0	0	0	0	1

Conformément à la procédure de démarrage à ce stade et durant 5 secondes on ne peut qu'arrêter la rotation droite du tapis B. Seul D9 est disponible.

b/ Recensement des événements

Les événements susceptibles de modifier l'ensemble des services disponibles (c.a.d de modifier le comportement) sont :

Ev-3 : Une exécution de D9, pour revenir à l'étape 1

Ev-5 : L'écoulement d'une temporisation de 5 seconde pour aller à une étape 4

En explorant les différentes étapes de fonctionnement qui apparaissent, jusqu'à revenir à des étapes existantes, on finit par obtenir le graphe des menus suivant :

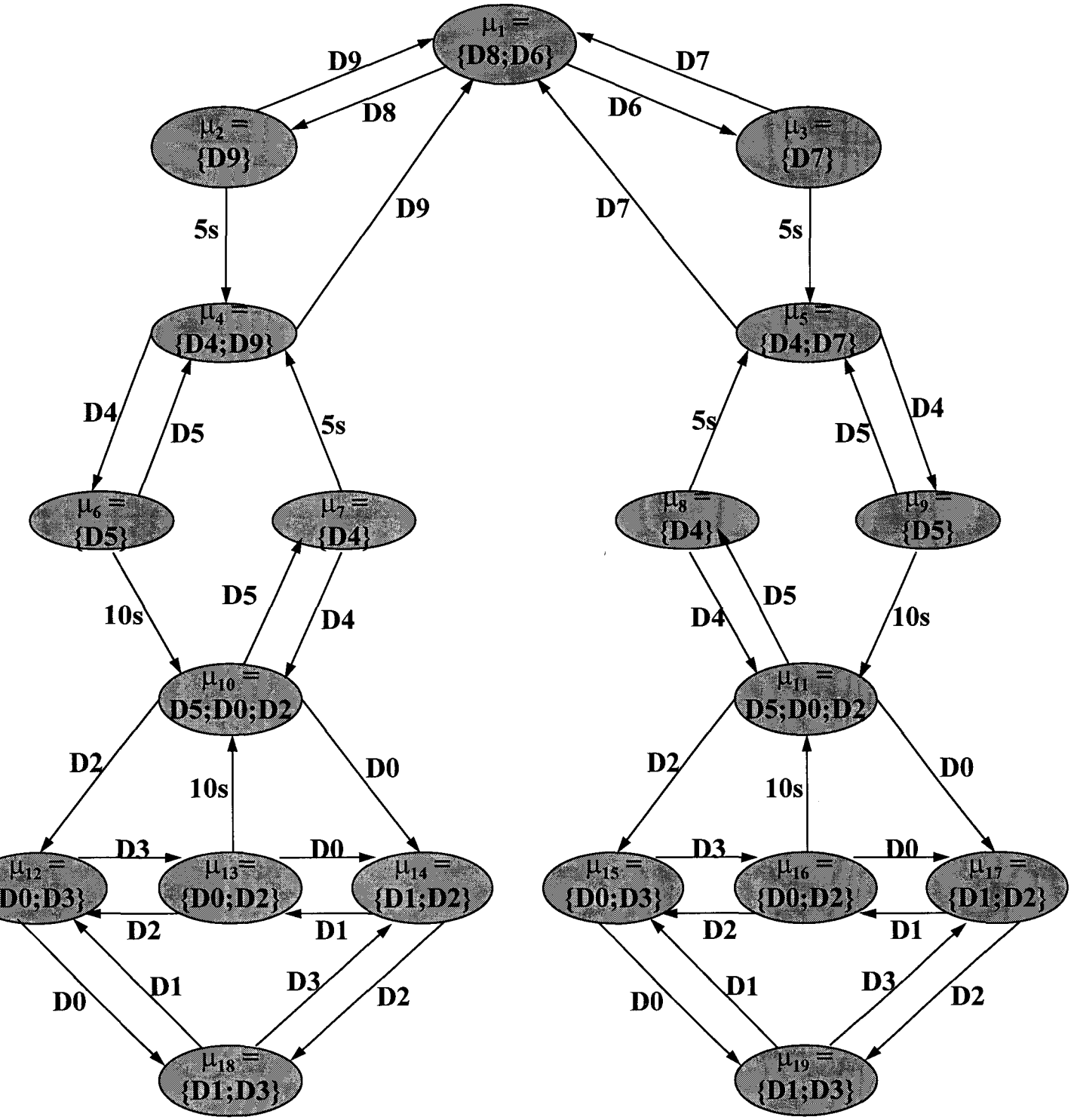


Figure 6.3 : graphe des menus du mode d'utilisation manuelle de l'installation de manutention

6.5. Partition de l'ensemble des menus en classes d'équivalences : Simplification du graphe des menus.

6.5.1. Position du problème

Comme prévu dans le chapitre 4, des étapes différentes peuvent avoir des menus identiques. Pour un utilisateur extérieur les menus μ_{10} et μ_{11} de l'exemple sont équivalents. La solution proposée [§4.4] est de fusionner ces menus pour n'en faire qu'un dans le graphe des menus, ce dernier se transforme donc en graphe de classes de menus.

6.5.2. Agrégation des menus en classes d'équivalence.

D'après le graphe des menus du mode d'utilisation manuelle de l'installation de manutention, on obtient la matrice d'incidence représentée par le tableau des événements de changement de menus suivant :

	μ_1	μ_2	μ_3	μ_4	μ_5	μ_6	μ_7	μ_8	μ_9	μ_{10}	μ_{11}	μ_{12}	μ_{13}	μ_{14}	μ_{15}	μ_{16}	μ_{17}	μ_{18}	μ_{19}
μ_1		D8	D6																
μ_2	D9			5s															
μ_3	D7				5s														
μ_4	D9					D4													
μ_5	D7								D4										
μ_6				D5						10s									
μ_7				5s						D4									
μ_8					5s						D4								
μ_9					D5						10s								
μ_{10}							D5					D2		D0					
μ_{11}								D5							D2		D0		
μ_{12}													D3					D0	
μ_{13}										10s		D2		D1					
μ_{14}													D1					D2	
μ_{15}																D3			D0
μ_{16}											10s				D2		D0		
μ_{17}																D1			D2
μ_{18}												D1		D3					
μ_{19}															D1		D3		

L'agrégation des menus se fait par partition en classes d'équivalence. Nous désignons chaque classe par la suite croissante des numéros des services qu'elle propose, exemple : la classe 025 propose les services {D0 ; D2 ; D5}. La partition en classes d'équivalences (sans

levée d'indéterminisme) permet d'obtenir le tableau des événements de changement de classes de menus suivant :

	68	7	9	49	47	4	5	025	03	02	12	13
68	x	D6	D8	x	x	x	x	x	x	x	x	x
7	D7	x	x	x	5s	x	x	x	x	x	x	x
9	D9	x	x	5s	x	x	x	x	x	x	x	x
49	D9	x	x	x	x	x	D4	x	x	x	x	x
47	D7	x	x	x	x	x	D4	x	x	x	x	x
4	x	x	x	5s	5s	x	x	4	x	x	x	x
5	x	x	x	D5	D5	x	x	10s	x	x	x	x
025	x	x	x	x	x	D5	x	x	D2	x	D0	x
03	x	x	x	x	x	x	x	x	x	D5	x	D0
02	x	x	x	x	x	x	x	10s	D2	x	D0	x
12	x	x	x	x	x	x	x	x	x	D1	x	D2
13	x	x	x	x	x	x	x	x	D1	x	D3	x

Ce dernier tableau permet de voir rapidement où se posent les indéterminismes. Si on observe bien ce tableau, on se rend compte que les seuls indéterminismes qui existent concernent les classes 4 et 5. En effet il apparaît que 5 secondes après avoir été dans la classe 4 on ne sait pas s'il faut aller à la classe de menus 49 ou à 47. De même pour la classe 5, après une exécution de D5 on ne sait pas s'il faut aller à la classe de menus 49 ou à 47.

Les indéterminisme posés dans le paragraphe précédent sont dus [ch4 proposition 4.8] aux transitions suivantes du graphe des menus :

$(\mu_6 ; D5 ; \mu_4)$ et $(\mu_9 ; D5 ; \mu_5)$: μ_6 est équivalent à μ_9 alors que μ_4 n'est pas équivalent à μ_5

$(\mu_7 ; 5s ; \mu_4)$ et $(\mu_8 ; 5s ; \mu_5)$: μ_7 est équivalent à μ_8 alors que μ_4 n'est pas équivalent à μ_5

Le tableau indique aussi que pour les classes de menus 13 ; 12 ; 03 ; 02, il n'y a pas d'indéterminisme, ce résultat est prévisible, il est du au fait que pour chacune de ces classes, les différentes transitions sortantes de leurs menus ne posent pas d'indéterminisme. Par exemple, la classe 12 est obtenue en associant les menus μ_{14} et μ_{17}, ces derniers ont chacun une transition sortante conditionnée par D1 pour aller à μ_{13} (cas de μ_{14}) ou à μ_{16} (cas de μ_{17}). Cette situation aurait créé un indéterminisme si les menus μ_{13} et μ_{16} n'avaient pas été équivalents.

6.5.3. Levée des indéterminismes

6.5.3.1. Levée de l'indéterminisme concernant la classe "5"

D'après le graphe des menus (figure 2), la classe 5 est obtenue en associant les menus μ_6 et μ_9. Au sens de la définition 11 [ch4 définition 4.11] les transitions (μ_6 ; D5 ; μ_4) et (μ_9 ; D5 ; μ_5) posent alors des indéterminismes. Cette situation est due au fait que μ_6 et μ_9 sont équivalentes alors que μ_4 et μ_5 ne le sont pas. Ainsi en fusionnant les sommets μ_6 et μ_9 du graphe des contraintes on ne peut pas savoir si une exécution de D5 nous fait passer de ce nouveau sommet vers la classe de μ_4 ou vers celle de μ_5.

D'après le graphe des contraintes, pour accéder à la classe 5, il faut exécuter l'une des deux séquences suivantes:

D8 puis 5s puis D4 (pour atteindre μ_6)

ou D6 puis 5s puis D4 (pour atteindre μ_9)

Ainsi [Déf 16 §4.3.2.2], afin de savoir par où on a accédé à la classe de menu 5 il suffit alors d'évaluer les variables binaires $\omega(\mu_6)$ et $\omega(\mu_9)$. Dans notre cas on a :

$\omega(\mu_6)$ = D8 / 5s / D4

et $\omega(\mu_9)$ = D6 / 5s / D4

Comme à chaque fois qu'on se trouve dans la classe 5 la séquence [5s/ D4] s'est forcément produite, pour lever les indéterminismes on peut se contenter des variables :

$\omega(\mu_6)$ = D8

et $\omega(\mu_9)$ = D6

Ainsi, d'après la proposition 4.9 [ch 4], on lève les indéterminismes en conditionnant le passage de la classe 5 à la classe 49 par l'événement [D5 / D8] et non plus par [D5] uniquement, par contre le passage de 5 à 47 est conditionné par [D5 / D6].

6.5.3.2. Levée de l'indéterminisme concernant la classe "4"

Cet indéterminisme est du aux transitions suivantes :

(μ_7 ; 5s ; μ_4) et (μ_8 ; 5s ; μ_5) : μ_7 est équivalent à μ_8 alors que μ_4 n'est pas équivalent à μ_5

D'après le graphe des contraintes, pour accéder à la classe 4 il faut exécuter les séquences :

D8 puis 5 secondes puis D4 puis 10 secondes puis D5 (pour atteindre μ_7)

ou D6 puis 5 secondes puis D4 puis 10 secondes puis D5 (pour atteindre μ_9)

En appliquant la même démarche que dans le cas de la classe 5, on se rend compte que pour lever les indéterminismes concernant la classe 4, il suffit de conditionner, dans le graphe des classes de menus, le passage de la classe 4 à la classe 49 par l'événement [5s / D8], par contre le passage de 4 à 47 est conditionné par [5s / D6].

6.5.4. Graphe des classes de menus et répercussion des contraintes

Une fois que tous les indéterminismes dus à l'agrégation des menus en classes d'équivalence ont été levés nous obtenons le graphe des classes de menus suivant (figure 3).

6.5.5. Graphe des classes de menus

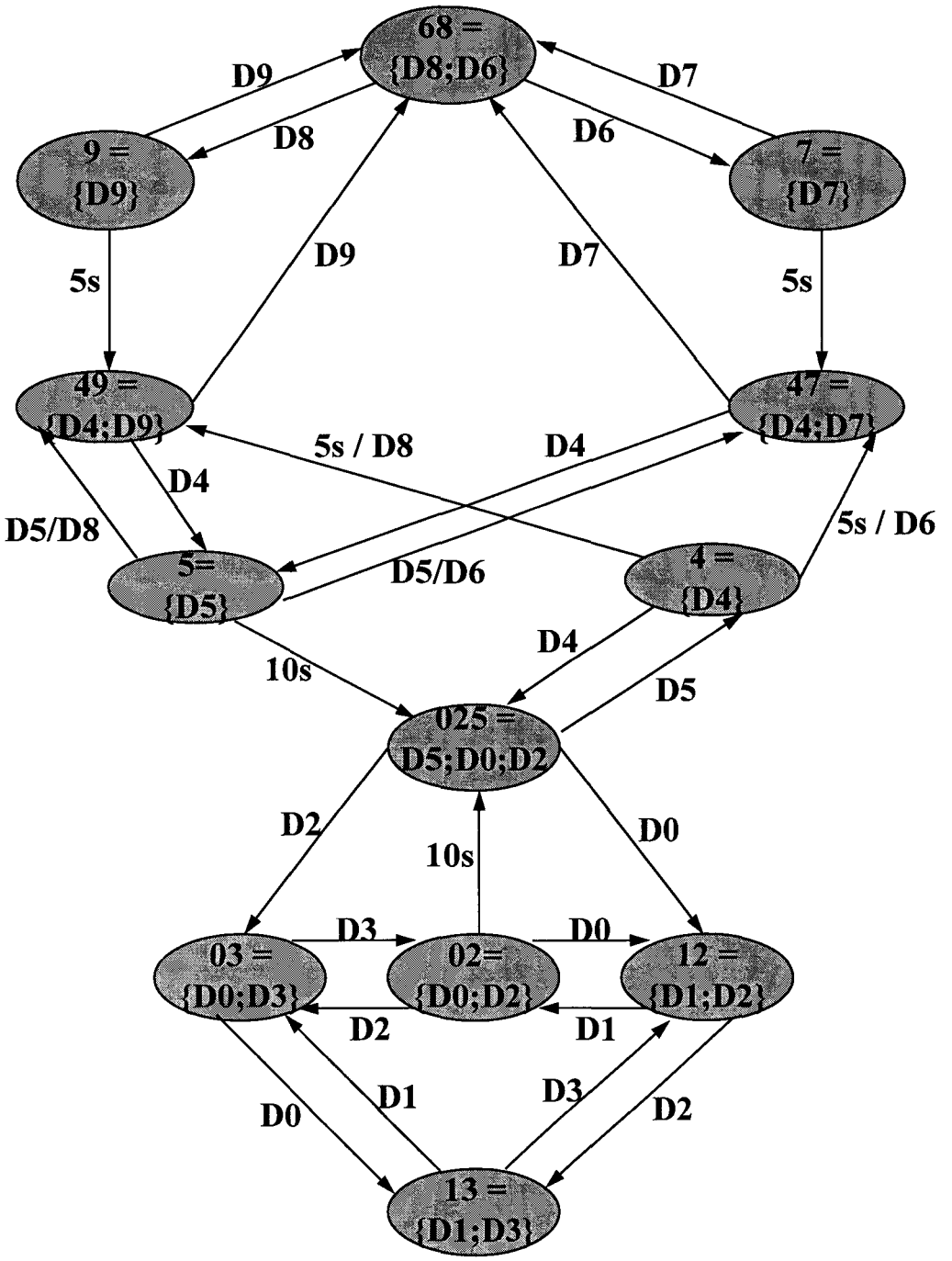


Figure 6.4 : graphe des classes menus de l'installation de manutention

6.5.6. Répercussion des contraintes sur chaque équipement

Si on imagine que chaque instrument est commandé par une entité de commande indépendante, pour obtenir le comportement global souhaité de l'ensemble de l'installation il faut organiser les services de chaque instrument de la façon suivante :

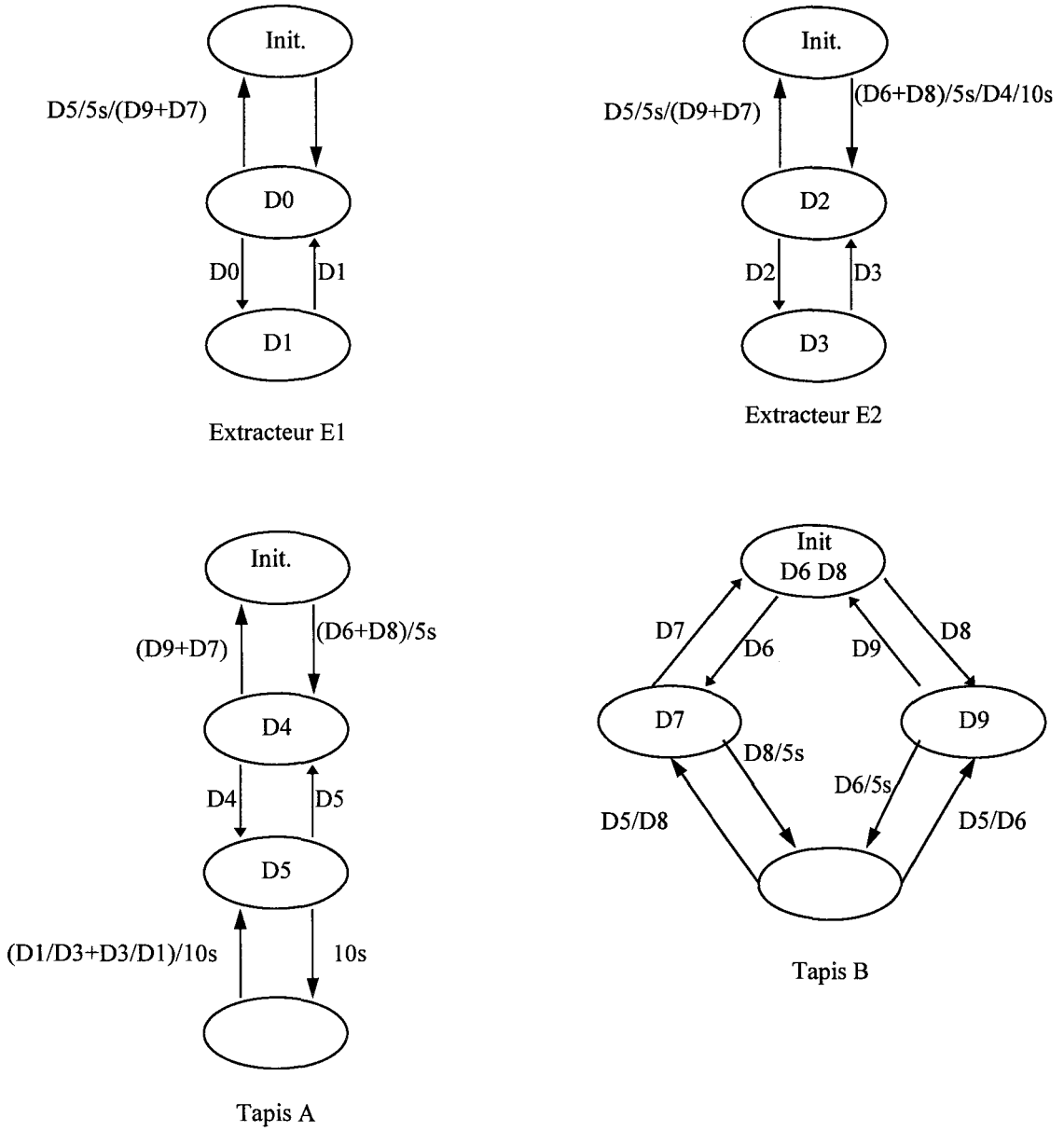


Figure 6.5 : Répercussion des contraintes sur chaque équipement

Le graphe de menus de chaque instrument est obtenu tout d'abord en éliminant du graphe des menus de l'ensemble de l'installation les services qui ne sont pas proposés par l'équipement considéré, ensuite en regroupant en un seul sommet les menus identiques.

6.5.7. Représentation grafcet du graphe des menus.

Le GRAFCET (GRAphe Fonctionnel de Commande Etat Transition) est un modèle formel de représentation graphique permettant de décrire une logique séquentielle relative au fonctionnement d'un automate ou de tout autre système à entrées et sorties booléennes.

Dans ce paragraphe, nous nous proposons de représenter le graphe des menus obtenu dans [§6.5.5] dans les normes du grafcet. Nous obtenons de cette façon le grafcet représenté par la figure 6.6 :

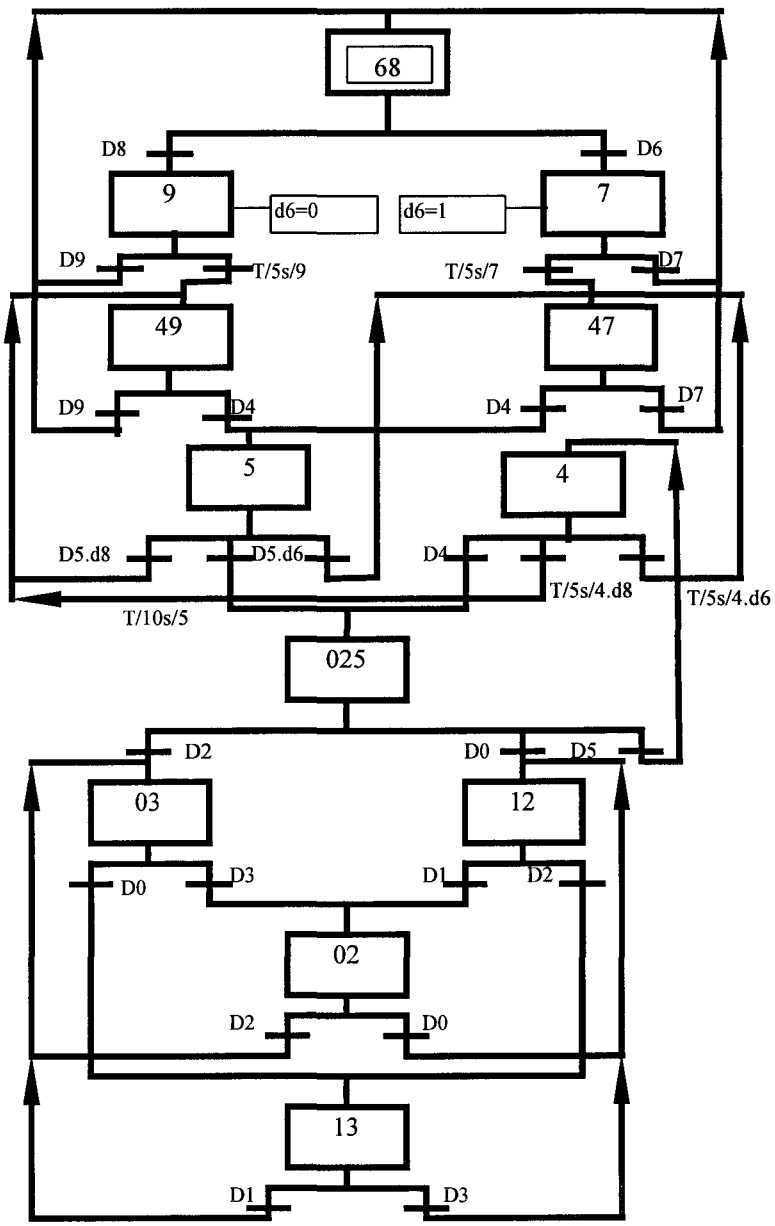


Figure 6.6 : grafcet du système

Par le fait que le graphe des menus est unique, la présente démarche nous permet d'aboutir, dans tous les cas, au même grafcet représentant la commande du système. Toutefois, le grafcet obtenu ne pourra contenir ni des "ET" logiques ni des "OU" non exclusifs. En effet ces dernières restrictions traduisent les faits que:

. premièrement, il n'est pas possible de proposer à l'utilisateur deux menus différents au même moment. Ainsi, il n'est pas possible de valider deux étapes différentes par la même réceptivité.

. Deuxièmement, nous avons postulé dans [§4.3.5] que la granularité du temps était suffisamment faible pour supposer que des événements indépendants ne peuvent pas se produire en même temps.

6.6. Automatisation du processus :

6.6.1. Position du problème:

A ce stade de leur étude, les auteurs de l'exemple se proposent de décrire l'automatisation du processus de manutention. Ils décident ainsi d'automatiser les séquences de démarrage et d'arrêt de l'installation. Dans cette partie, nous allons montrer comment la démarche intuitive des auteurs peut être remplacée par une démarche formelle. Ainsi nous allons retrouver l'expression de toutes les séquences de conduite automatisables par le processus.

6.6.2. Langage des séquences automatisables

6.6.2.1. Matrice d'incidence du sous graphe des menus dépendant de l'utilisateur

Dans le cas précis de cet exemple, il n'existe pas d'événement indépendants de l'utilisateur. En effet, l'occurrence de chaque événement dépend uniquement d'un mot de commande que doit préalablement formuler l'utilisateur.

ainsi la matrice d'incidence du sous-graphe des menus dépendant de l'utilisateur de l'exemple choisi est la matrice « A » suivante :

	68	7	9	49	47	4	5	025	03	02	12	13
68	x	D6	D8	x	x	x	x	x	x	x	x	x
7	D7	x	x	x	5s	x	x	x	x	x	x	x
9	D9	x	x	5s	x	x	x	x	x	x	x	x
49	D9	x	x	x	x	x	D4	x	x	x	x	x
47	D7	x	x	x	x	x	D4	x	x	x	x	x
4	x	x	x	5s/D8	5s/D6	x	x	4	x	x	x	x
5	x	x	x	D5/D8	D5/D8	x	x	10s	x	x	x	x
025	x	x	x	x	x	D5	x	x	D2	x	D0	x
03	x	x	x	x	x	x	x	x	x	D5	x	D0
02	x	x	x	x	x	x	x	10s	D2	x	D0	x
12	x	x	x	x	x	x	x	x	x	D1	x	D2
13	x	x	x	x	x	x	x	x	D1	x	D3	x

6.6.2.2. Vecteur des langages propres aux menus

Rappelons que ce vecteur définit pour chaque menu le langage des commandes que peut formuler l'utilisateur sans pour autant quitter le menu en question. Pour l'exemple choisi ce vecteur « L_μ »est le suivant :

L_μ	$L_{68}=T_\infty$
	$L_7=T_{<5s}$
	$L_9=T_{<5s}$
	$L_{49}=T_\infty$
	$L_{47}=T_\infty$
	$L_4=T_{<5s}$
	$L_5=T_{<10s}$
	$L025=T_\infty$
	$L_{03}=T_\infty$
	$L_{02}=T_{<10s}$
	$L_{12}=T_\infty$
	$L_{13}=T_\infty$

Remarque :

Certains menus permettent théoriquement à l'utilisateur d'être indéfiniment dans l'étape correspondante. En effet pour les menus : 68 ;49 ;47 ;025 ;12 ;13 il n'existe pas d'événement indépendant de l'utilisateur donc l'occurrence nous oblige à quitter l'un de ces menus. Ces derniers peuvent être qualifiés de stables.

6.6.2.3. Vecteur des chemins sortants de chaque menu

Ce vecteur à été défini dans [§5.5.1.2 Déf5.9], il vérifie l'équation Ls = A.Ls+B. La solution est donc le produit matriciel : Ls=(A*).B

L'obtention de la matrice A* est un problème délicat qui a fait l'objet de différents travaux de recherche. Les principaux algorithmes sont ceux de Jacobi, Gauss-Seidel et Jordan. Pour plus de détail nous renvoyons le lecteur aux travaux de Gondran et plus précisément vers [pp83 §Algorithmes généraux cf algèbre des chemin Graphe et algorithmes].

L'application de ces algorithmes nous permet d'obtenir le vecteur Ls. Le résultat final est présenté dans le paragraphe suivant.

6.6.2.4. Forme générale des expressions d'un mot du langage des séquences automatisables de l'installation choisie comme exemple

Nous pouvons à ce stade, définir formellement l'expression générale des mots du langage de commande que l'utilisateur est autorisé à formuler pour la conduite de l'installation de manutention choisie comme exemple. En effet ce langage vérifie $L= Ls^T.L_\mu = B^T.A^*. L_\mu$ Ainsi nous obtenons :

L'expression la plus générale d'un mot du langage de commande de l'installation de manutention est :

$$
\begin{aligned}
&[\\
&((T\ D8\ T_{<5s}\ D9)^*D8\ T_{\geq 5s}\ D9\ T)^*\ D8\ T_{\geq 5s}\\
&((D4\ T_{<10s}\ D5\ T)^*D4\ T_{\geq 10s}\ (D5\ T_{<5s}\ D4\ T)^*D5\ T_{\geq 5s}\ )^*\ D4\ T_{\geq 10s}\\
&\{D2\ T\ ((D3\ T_{<10s}\ D2\ T)^*\ T_{<10s}\ (D0\ T\ D1\ T_{<10s})^*\ D0\ T\ (D2\ T\ D3\ T)^*\ D2\ T\ (D1\ T\ D0\ T)^*\ )^*\ [D3\\
&T_{\geq 10s} + (D3\ T_{<10s}\ D2\ T)^*\ T_{<10s}\ (D0\ T\ D1\ T_{<10s})^*D1\ T_{\geq 10s}\ ]\}^*\\
&\{D0\ T\ ((D1\ T_{<10s}\ D0\ T)^*\ T_{<10s}\ (D2\ T\ D3\ T_{<10s})^*\ D2\ T\ (D0\ T\ D1\ T)^*\ D0\ T\ (D3\ T\ D2\ T)^*)^*\ [D1\ T_{\geq 10s}\\
&+ (D1\ T_{<10s}\ D0\ T)^*\ T_{<10s}\ (D2\ T\ D3\ T_{<10s})^*D3\ T_{>10s}\ ]\}^*\\
&(D5\ T_{<5s}\ D4)\ D5\ T_{\geq 5s}\ (D4\ T_{<10s}\ D5)^*\ D4\ T_{\geq 10s})^*\ D5\ T_{\geq 5s}\ (D4\ T_{<10s}\ D5)^*\ D9\\
&+\\
&((T\ D6\ T_{<5s}\ D7)^*\ D6\ T_{\geq 5s}\ D7\ T)^*D6\ T_{\geq 5s}\\
&((D4\ T_{<10s}\ D5\ T)^*D4\ T_{\geq 10s}\ (D5\ T_{<5s}\ D4\ T)^*D5\ T_{\geq 5s}\ )^*\ D4\ T_{\geq 10s}\\
&\{D2\ T\ ((D3\ T_{<10s}\ D2\ T)^*\ T_{<10s}\ (D0\ T\ D1\ T_{<10s})^*\ D0\ T\ (D2\ T\ D3\ T)^*\ D2\ T\ (D1\ T\ D0\ T)^*\ )^*\ [D3\\
&T_{\geq 10s} + (D3\ T_{<10s}\ D2\ T)^*\ T_{<10s}\ (D0\ T\ D1\ T_{<10s})^*D1\ T_{\geq 10s}\ ]\}^*\\
&\{D0\ T\ ((D1\ T_{<10s}\ D0\ T)^*\ T_{<10s}\ (D2\ T\ D3\ T_{<10s})^*\ D2\ T\ (D0\ T\ D1\ T)^*\ D0\ T\ (D3\ T\ D2\ T)^*)^*\ [D1\ T_{\geq 10s}\\
&+ (D1\ T_{<10s}\ D0\ T)^*\ T_{<10s}\ (D2\ T\ D3\ T_{<10s})^*D3\ T_{>10s}\ ]\}^*\\
&(D5\ T_{<5s}\ D4)\ D5\ T_{\geq 5s}\ (D4\ T_{<10s}\ D5)^*\ D4\ T_{\geq 10s})^*\ D5\ T_{\geq 5s}\ (D4\ T_{<10s}\ D5)^*\ D7\\
&]^*
\end{aligned}
$$

6.6.3. Automatisation de quelques séquences

6.6.3.1. Séquence de démarrage et d'arrêt des tapis :

Supposons que le but du système d'automatisation du processus présenté ici soit de prendre en charge les séquences de démarrage et d'arrêt des tapis. Pour l'analyse décisionnelle, le système a une nouvelle configuration dont la distribution des décisions est la suivante.

Les décisions D0 à D3 restent toujours à la charge de l'environnement.

Les décisions D4 à D10 sont déléguées au système.

L'environnement est chargé des nouvelles décisions:

D11 : Démarrage de l'ensemble des tapis vers silo S1

D12 : Démarrage de l'ensemble des tapis vers silo S2

D13 : arrêt de l'ensemble des tapis

Bien que les auteurs de l'exemple expriment l'objectif de l'automatisation, ils résolvent le problème uniquement par la création de nouvelles décisions. Cependant, on ne sait pas comment D11, D12 et D13 sont réalisés à partir de D4, D5, D6, D7, D8, D9 et D10.

Pour prévoir d'automatiser une séquence il faut d'abord définir à quoi elle correspond exactement, ensuite il faut vérifier que celle-ci est autorisée par l'équipement. Dans le cas de notre modèle de description, il suffit de vérifier que celle-ci correspond à un chemin dans le graphe des menus.

La procédure de démarrage des tapis vers le silo S1 correspond à une exécution de D8, suivie d'une attente de 5 secondes, suivie d'une exécution de D4 et enfin d'une attente de 10 secondes soit : D8/ 5s/ D4/ 10s.

Cette séquence correspond en fait au chemin dans le graphe des classes de menus qui nous fait passer de la classe 68 à la classe 025. C'est donc une séquence réalisable par l'équipement, on peut la définir comme un nouveau service dit composé, soit donc:

D11 = D8/ 5s/D4/ 10s

On vérifie de même que pour D12 et pour D13 on a :

D12 = D6/ 5s/ D4/ 10s

D13 = D5/ 5s/ D9/ s'il y a eu D8 ou D5/ 5s/ D7 s'il y a eu D6

6.6.3.2. Paramètrage du sens de rotation des tapis

Dans la suite, les auteurs remarquent que les décisions D11 et D12 peuvent être remplacées par une seule décision (D'11) avec une consigne qui paramétrise le sens de rotation du tapis B. Dans notre modèle D'11 définit un nouveau service qui n'est réalisable que si D11 et D12 appartiennent au même menu (ce qui est le cas).

Les nouveaux services ainsi définis permettent une commande plus élaborée. Ces services s'ajoutent à ceux déjà existants. Un problème de redondance de commande peut se poser. Deux solutions sont possibles. La première est de garder la redondance et de proposer tous les services à l'utilisateur, dans ce cas on gagne en richesse de commande mais on perd en simplicité. Pour structurer les services, on peut les organiser en modes d'utilisation figure 6.7:

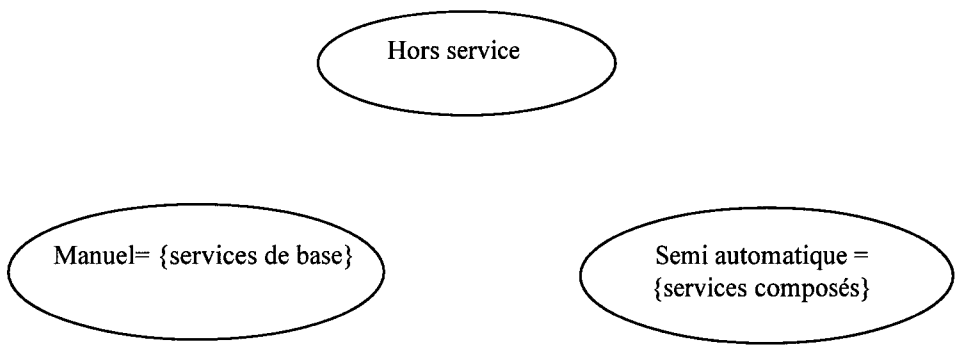


Figure 6.7 : organisation des modes d'utilisation

Il faut bien entendu créer de nouvelles requêtes de changement de mode.

La seconde solution est d'éliminer la redondance en ne gardant que les nouveaux services, cette solution se traduit par une modification du graphe des classes de menus. Dans le dernier cas on obtient les graphes des figures 6.8 et 6.9.

La définition des services composés présente un net avantage non seulement pour l'automatisation progressive d'un processus, mais aussi pour définir les différentes finalités que peut avoir l'utilisateur et qui sont réalisables par l'équipement.

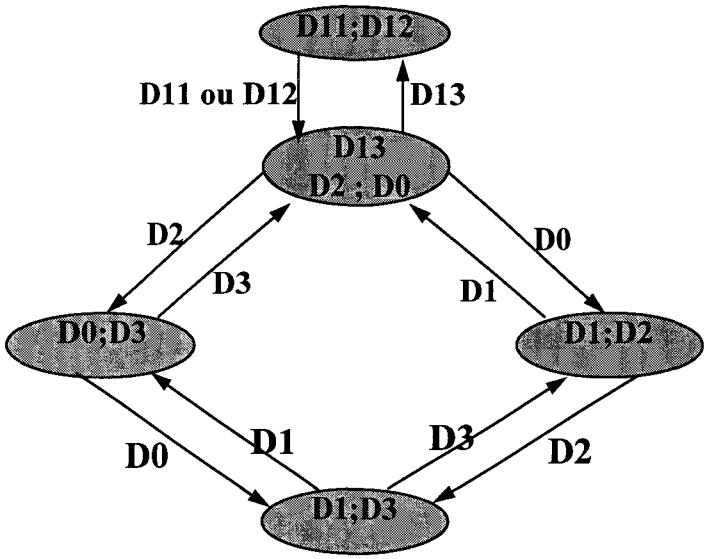


Figure 6.8 : mode d'utilisation semi-automatique

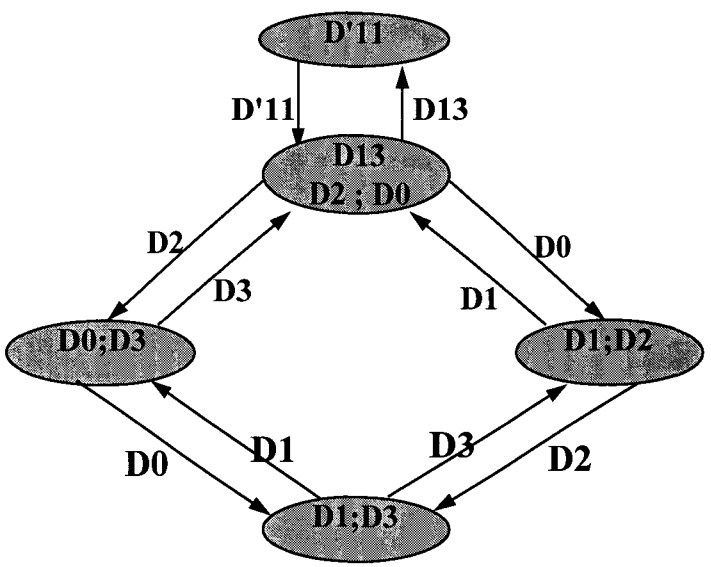


Figure 6.9 : mode d'utilisation semi-automatique avec paramétrage du sens de rotation

6.7. Conclusion

Dans ce chapitre, nous avons voulu illustrer par un exemple la façon dont s'appliquent les résultats obtenus dans les chapitres précédents à un cas concret. L'exemple, choisi dans la littérature, a été utilisé par ses auteurs pour illustrer la démarche d'automatisation par analyse décisionnelle.

Ainsi, nous avons dans un premier temps exprimé les contraintes relatives à l'installation de manutention choisie comme exemple, nous avons donc retrouvé le graphe des menus représentant les utilisations cohérentes des services offerts par les différents équipements de l'installation. Ensuite, nous avons exprimé formellement l'ensemble des séquences d'exécution de service pouvant être automatisées. Le but était de remplacer la démarche des auteurs de l'exemple qui relève d'avantage de l'expérience par une démarche formelle elle-même automatisable.

7. Conclusion et Perspectives

Les travaux effectués jusqu'à présent sur le thème de l'instrumentation intelligente se sont intéressés à la nature des structures internes et des traitements implantés dans un instrument. Ces travaux s'adressent donc particulièrement aux concepteurs. Les utilisateurs, quant à eux, manifestent d'autres besoins et se préoccupent peu de la nature des structures internes. Dans ce mémoire, nous avons voulu définir une approche générale indépendante des structures internes et qui permet de modéliser la vision que peuvent avoir les utilisateurs d'un instrument intelligent. Il s'agit donc de formaliser une approche externe (ou utilisateur) dans le concept de l'instrumentation intelligente.

Dans un premier temps, notre but était de disposer d'une modélisation descriptive des potentialités techniques offertes par un instrument intelligent. Nous avons ainsi défini dans le chapitre 2 un modèle générique à partir duquel il est possible de décliner la description particulière d'un instrument intelligent tel qu'il est vu par un utilisateur externe ou à travers un réseau de communication. Nous avons appelé ce modèle générique modèle externe de spécification d'un instrument intelligent. Le modèle externe définit un instrument comme une entité proposant des services, lesquels manipulent des variables et font appel à un ensemble de ressources. Toutefois, les services sont organisés en sous ensembles cohérents dits modes d'utilisation. Cette organisation joue le rôle d'un premier filtre de commande pour éviter que l'utilisateur réalise des actions incompatibles. L'organisation des services en modes d'utilisation et l'identification des différentes situations de défaillance, nous a permis de définir formellement la notion de mode de marche d'un instrument intelligent.

Par la suite, nous nous sommes rendu compte que la notion de mode d'utilisation introduite dans le chapitre 2 n'était plus un filtre de commande suffisant lorsqu'il s'agissait d'analyser la conduite d'un ensemble d'instruments dans le but de réaliser une application particulière. En effet, si l'on part du principe que la conduite d'un ensemble d'instruments intelligents

s'inscrit toujours dans le contexte où les services proposés sont utilisés pour satisfaire une finalité précise, on s'aperçoit que les possibilités d'une coopération entre les éléments d'un ensemble d'instruments peuvent dépendre de la finalité que l'on souhaite réaliser. Il convenait donc, et c'était l'objet des chapitres 3 et 4, de spécifier les contraintes de conduite d'une application en relation avec le modèle externe décrit dans le chapitre 2. Pour résoudre le problème, nous avons, dans un premier temps, établi un lien entre la notion de contrainte et celle d'événement. Ensuite, nous avons constaté que durant un intervalle de temps encadré par l'occurrence de deux événements l'ensemble des services activables par l'utilisateur ne varie pas. Nous avons appelé ce type d'intervalle de temps "étape de fonctionnement" et l'ensemble des services activables "menu d'utilisation". Pour finir, nous avons démontré que les menus et les événements peuvent être structurés dans un graphe états transitions que nous appelons graphe des menus. Ce dernier permet d'exprimer la répercussion au niveau de l'utilisateur des restrictions de conduite que peut imposer la finalité d'une application répartie construite en interconnectant des instruments intelligents.

A partir du moment où nous avons spécifié la conduite à adopter pour réaliser une application particulière, nous avons envisagé d'étudier la possibilité de déléguer des séquences de conduite d'une application à un système d'automatisation. L'objectif était de définir des niveaux de commande supérieurs permettant une commande plus élaborée. Nous avons résolu le problème dans le chapitre 5 grâce à la théorie des langages et à l'algèbre des chemins de la théorie des graphes. Ces deux derniers outils nous ont permis de retrouver l'expression la plus générale d'une séquence d'exécution de services pouvant être déléguées à un système d'automatisation.

Dans le dernier chapitre de ce mémoire, nous avons illustré par un exemple de la littérature l'application des différentes notions introduites dans les chapitres précédents, l'exemple choisi a été utilisé dans plusieurs articles pour illustrer d'autres démarches d'automatisation.

Dans les perspectives de ce travail, nous voyons principalement trois volets:

1 Le premier se baserait sur la théorie des graphes et concernerait la formalisation des critères d'interopérabilité de conduite d'instruments intelligents. Cette perspective est possible grâce au fait que nous avons pu spécifier du point de vue de la conduite, par des entités de même nature, aussi bien l'architecture support construite en interconnectant un ensemble d'instruments que l'architecture

fonctionnelle d'une application. En effet, nous avons d'une part le graphe des modes d'utilisation pour spécifier les possibilités de conduite autorisées par un ensemble d'instruments, d'autre part, le graphe des menus pour spécifier les restrictions de conduite imposées par la finalité d'une application. Il convient donc de formaliser les possibilités de projections du graphe des menus sur celui des modes d'utilisation ceci afin de prévoir l'interopérabilité de conduite d'un ensemble d'instruments dans le cadre d'une application. Ce travail consisterait à généraliser la démarche présentée dans [§6.5.6].

2 Le troisième volet est de réaliser un atelier logiciel permettant de manipuler les différentes notions introduites dans le présent mémoire. Ce troisième volet ne pouvait être possible qu'après avoir spécifié les principales entités qui peuvent caractériser l'instrumentation intelligente.

ANNEXE 1

Proposition

Soit S un ensemble fini et dénombrable de services tel que Card(S)=N

Soient S^+ et S^- des sous-ensembles de S.

Alors, le nombre de couples (S^+,S^-) tels que $S^+ \cap S^- = \varnothing$ est égal à : 3^N

Démonstration

1 le nombre de sous ensemble de S est :

$$\text{Card}(P(S)) = \sum_{k=0}^{k=N} C_N^k = (1+1)^N = 2^N ,$$

2 cherchons d'abord le nombre de couples (S^+,S^-) tels que $S^+ \cap S^- = \varnothing$ et tels que S^+ possède i éléments:

- il existe C_N^i S^+ ayant i éléments
- pour chaque S^+, les S^- doivent être prix dans un ensemble à N-i éléments il existe donc 2^{N-i} S^- possibles.

Ainsi le nombre de couples (S^+,S^-) tels que $S^+ \cap S^- = \varnothing$ et S^+ possède i éléments est égal à : $2^{N-i} \times C_N^i$.

3 conclusion, le nombre total de couple (S^+,S^-) tels que $S^+ \cap S^- = \varnothing$, est égal à :

$$\sum_{i=0}^{i=N} (2^{N-i} . C_N^i) = (1+2)^N = 3^N$$

BIBLIOGRAPHIE

[AFNO 92] C 46-645, Normalisation Française, AFNOR (1992)

[AKAI 96] J. Akaichi, "Système automatisé de production à intelligence distribuée", Thèse de Doctorat, Université de Lille I, 1996.

[ARNO 92] A. Arnold, I Guessarian, "Mathèmatique pour l'informatique", Edition Masson 1992.

[BAYA 95 a] M. Bayart, "Architecture des systèmes automatisés de production à architecture distribuée", Groupement de Recherche Automatique, Journées d'Etude SAPID Paris 1995.

[BAYA 92 b] M. Bayart, M Staroswiecki, "Smart actuators : generic functional architecture, service and cost analysis", in Proc IEEE SICICI 92, Singapour February 1992.

[BAYA 92 c] M. Bayart, M Staroswiecki, "A generic functional model of smart instrument for distributed architectures", IMEKO International Symp. On Intelligent Instruments For Remote and On-site Measurements, Bruxelles, May 1993.

[BENZ 91] CI. Benzaken, "Systèmes formels : introduction à la théorie des langages", Masson 1991.

[BILA 94] P. Biland, "Modélisation des modes de marche d'un système automatisé de production", Thèse de Doctorat de l'Ecole Centrale de Nantes, 1994.

[BOIS 91] S. Bois, "Intégration de la gestion des modes de marche dans le pilotage d'un système automatisé de production", Thèse de Doctorat, Université de Lille I, 1991.

[BOUR 95] A. Bouras, M. Bayart, M. Staroswiecki, "Specification of smart instruments for intelligent control", IEEE SMC, Vancouver, Canada, 22-25 October 1995.

[BUCK 94] J. Bucki, Y. Pesqueux, "Modes de marche des systèmes automatisés", Revue française de gestion industrielle, N°1, 1994.

[BUCK 91 a] J. Bucki, Y. Pesqueux, L. Lasoudis, "B-COD: La conception orientée objets décisionnels", Groupe HEC, 78351 Jouy-en Josas cedex France, 1991.

[BUCK 91 b] J. Bucki, Y. Pesqueux, "Organe décisionnel et controle-délégation et automatisation", CR 388/1991, Groupe HEC, Les cahiers de recherche, Chambre de Commerce et d'Industrie de Paris.

[BUCK 91 c] J. Bucki, Y. Pesqueux, "Systéme d'information", Groupe HEC, 78351, Jouy-en Josas, France, 1991.

[CASS 92] J.P Cassar, M. Bayart, M. Staroswiecki, "Hierarchical data validation in control systems using smart actuators and sensors", In Proc. IFAC SICICA 92, Malaga, May 1992, pp 209-214.

[CHUN 67] K.L. Chung, "Markov chains with stationnary transition probability", Springer-Verlag, Berlin Heidelberg, New York, 1967.

[COCQ 93] V. Cocquempot, "Surveillance des processus industriels complexes, génération et optimisation des relations de redondances analytiques", Thèse de doctorat, Université de Lille I, 1993.

[CULL 75] G. Cullmann, " Initiation aux chaînes de Markov", Masson, 1975.

[ELKH 93] S. Elkhatabi, "Intégration de la surveillance de bas niveau dans la conception des systèmes à événements discrets", Thèse de Doctorat, Université de Lille I, 1993.

[GEHI 94] A. L Gehin, "Analyse fonctionnelle et modèle générique des capteurs intelligents - application à la surveillance de l'anesthésie", Thèse de Doctorat, Université de Lille I, 1994.

[GODO 96] A. Godon, "Contribution à la commande des systèmes à événements discrets par réseaux de petri", Thèse de Doctorat de L'Ecole Doctorale de Nantes, 1996.

[GOND 79] M. Gondran, M. Minoux "Graphes et algorithmes", Eyrolles, 1979.

[LIST 83] A.M. Listier, "Principe fondamentaux des systèmes d'exploitations", EYROLLES 1983.

[LORE 94] P. Lorenz, "Le temps dans les architectures de communication :application au réseau de terrain FIP", Thèse de Doctorat, Institut Polytechnique de Lorraine, 1994.

[NAJA 92] N. Najafi "Smart sensors" IBM, Microelectronics Division, 1000 River Street, Essex Junction, Vermont USA,05452. 1992

[RIVE 96] J.P. Riviere, M. Bayart, J.M. Thiriet, A. Bouras, M. Robert, "Intelligent instruments. some modeling approaches", Measurement and control Vol 29 July/August 1996.

[SAHR 95] A. Sahraoui, "Expression des contraintes temporelles dans les architectures fonctionnelles", Groupement de Recherche Automatique, Journées d'Etude SAPID Paris 1995.

[SIMO 95] F. Simono-Lion, "La démarche de conception des systèmes d'automatisation", Groupement de Recherche Automatique, Journées d'Etude SAPID Paris 1995.

[STAR 96 a] M. Staroswiecki, M. Bayart, "Les actionneurs intelligents", Hermes 1996.

[STAR 96 b] M. Staroswiecki, M. Bayart, "Models and Languages for the Interoperability of Smart Instruments", Automatica, Vol 32, No. 6 pp 859-873, 1996.

[TRIC 97] S. Tricquet, A.L. Gehin, N. Devesa, M. Bayart, B. Toursel, "Fieldbus considered as a smart instrument : modelization using a generic external model", SICICA'97, Annecy, France, 9-11 juin 1997.

[VEGA 96] L. Véga "Modèle de coopération et de communication entre processus temps réel répartis", Thèse de Doctorat, Institut Nationale Polytechnique de Lorraine, septembre 1996.

[WEST 92] J. Westbrock, "Device description language specification", Document COM-20 de ROSMOUNT, 1992.

[WIEC 94] M.J. Wieczoreck, "Locative temporal logic and distributed real time systems", PhD, Proefschrift wiskunde en informatica nijmegen, 1994 (Found in web : univ-lille1.fr)

[WORD 96] EN50170, Interoperability WordFip 1996, WordFip field Bus, 2 rue Bône, 92160 Antony, France (web : wordfip.org)

[ZIMM 80] H. Zimmermann, OSI Reference Model, "The ISO model of architecture for open system interconnection", IEEE Trans. Commun. COM-28, 425-432, 1980

Table des matières

Première partie: Modèle générique pour l'interopérabilité d'instruments intelligents

Troisième partie: Automatisation de la conduite d'une application répartie construite en interconnectant des instruments intelligents