République algérienne démocratique et populaire
Ministère de l'enseignement supérieur et de la recherche scientifique
Université Mentouri – Constantine-
Faculté des sciences de l'ingénieur
Département d'informatique

Année : 2012
N° d'ordre :
Série :

Thèse

En vue de l'obtention du diplôme de Doctorat en sciences en informatique

Présentée par

MAAROUK TOUFIK MESSAOUD

Thème

Modèles formels pour la conception des systèmes temps réel

Soutenue le : 27 /06/2012 . Devant le jury composé de :

Pr Chaoui Allaoua	Président	Professeur	U.M.Constantine
Pr Saidouni Djamel Eddine	Rapporteur	Professeur	U.M.Constantine
Dr Merniz Salah	Examinateur	M.C.A	U.M.Constantine
Dr Boudour Rachid	Examinateur	M.C.A	U.BM.Annaba
Pr Kimour Mohammed Taher	Examinateur	Professeur	U.BM.Annaba

Thèse préparée au laboratoire MISC, Université Mentouri, 25000 Constantine, Algérie.

Modèles formels pour la conception des systèmes temps réel.

Toufik Messaoud MAAROUK

Equipe Conception Formelle des Systèmes Complexes
Laboratoire MISC, Université de Mentouri, 25000 Constantine

Remerciements

Je tiens tout d'abord à remercier Pr Chaoui Allaoua, professeur à l'université Mentouri -Constantine- qui a accepté la lourde tâche d'être président de jury.

Je remercie également Dr Merniz Salah, maitre de conférences à l'université Mentouri -Constantine-, Dr Boudour Rachid, maitre de conférences à l'université Badji Mokhtar -Annaba-, et Pr Kimour Mohammed Taher, professeur à l'université Badji Mokhtar -Annaba-, qui ont accepté d'être examinateurs de cette thèse, ce dont je suis extrêmement honoré.

Je tiens à exprimer ma gratitude envers mon directeur de thèse, Professeur Djamel Eddine SAIDOUNI pour toutes les discussions intéressantes, et la qualité de ses conseils. Il a été un directeur de thèse amical et disponible, notre travail en commun a été des plus agréables. Je lui en suis très reconnaissant.

Je remercie aussi tout les membres du laboratoire MISC car j'ai pu effectuer cette thèse dans un cadre très agréable.

Résumé

La conception des systèmes distribués, temps réel avec mobilité est une opération lourde en temps et en coût de développement. Ces systèmes sont dotés d'un comportement qui est contraint par le temps, c'est à dire doivent interagir correctement avec l'environnement non seulement au regard des informations échangées, mais également au regard des instants aux quels ces interactions se réalisent. Ces systèmes sont généralement hétérogènes du point de vue des composants matériels et logiciels, ils sont présents dans de nombreuses industries tel que : le contrôle de processus, les centrales nucléaires, l'avionique, le contrôle du trafic aérien, les télécommunicatons, les applications médicales, et les applications de défense. De plus, ces systèmes doivent satisfaire des contraintes dures pour assurer leur bon fonctionnement (respect des échéances temporelles. . .).

L'étude de ce type de systèmes fait appel à des méthodes formelles permettant de répondre aux exigences au quelles sont soumises. Dans la littérature, il a été proposé un grand nombre de techniques dotées d'un support mathématique pour raisonner sur la conformité des systèmes répartis. Dans ce contexte les algèbres de processus forment un outil mathématique pour raisonner sur le comportement des systèmes concurrents et communicants.

Ce travail de thèse exploite les différents modèles et langage de spécification des systèmes distribués, temps réel et mobiles. Nous proposons un modèle pour prendre en charge les aspects : distribution et temps. Nous s'étend par la suite ce dernier pour prendre l'aspect mobilité. Dans ce contexte, les sémantiques de vrai parallélisme, entre autre la sémantique de maximalité, conviennent à être employées lorsqu'on s'abstrait de l'hypothèse de l'atomicité temporelle et structurelle des actions. De ce fait, le modèle D-LOTOS[1], extension temporelle à l'algèbre de processus LOTOS[2], intégrant à la fois contraintes temporelles et durées des actions, sera étendu pour répondre aux aspects de mobilité et de distribution sous une sémantique de maximalité.

Mots-clés Sémantique de vrai parallélisme, Mobilité, Systèmes temps-réel, Systèmes distribués, LOTOS.

[1] Durational LOTOS

[2] Language Of Temporal Ordering Specification

Abstract

The design of distributed real-time system with mobility is a difficult operation in time and development cost. They have a behavior that is constrained by time, *ie* must successfully interact with the environment not only in terms of exchanged information, but also in terms of moments in which these interactions are accured. These systems are usually heterogeneous with both hardware and software components, they are present in many indusdries such as :process control, nuclear power plants, avionics, air traffic control, telecommunicatons, medical applications, and defense applications. In addition, these systems must meet hard constraints to ensure their smooth functioning (temporal deadlines...).

The study of such systems uses formal methods to meet the requirements to what is submitted. In the literature, it has been proposed many techniques based on support mathematical reasoning about the conformity of distributed systems. In this context process algebras are such mathematical tools for reasoning about the behaviour of concurrent and communicating systems.

This thesis exploits different models and languages for specifying distributed, real-time and mobile systems. Then, its proposes a model supporting : distribution and time. Then we extend it to take the mobility of processes. In this context, the semantics of true concurrency, like the maximality-based semantics, appropriate for abstracting temporal atomicity and structural actions assumption. The D-LOTOS language, temporal extension of process algebra LOTOS, incorporating both temporal and durations of actions. Will be extended for supporting both mobility, distribution under a maximality semantics.
Keywords True concurrency semantics, Mobility, Real-time systems, Distributed systems, LOTOS.

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Introduction générale

Sommaire

1.1 Contexte générale

Les systèmes temps réel distribués et mobiles sont devenus omniprésents, dans plusieurs domaines tels que les systèmes de contrôle de processus, les centrales nucléaires, l'avionique, le contrôle du trafic aérien, les télécommunications, les applications médicales, les technologies multimédias et les applications de défense. Ils sont généralement hétérogènes du point de vue composants matériels et logiciels ; ces systèmes sont de nature complexes.

La conception de tels systèmes nécessite des méthodes et outils de vérification formelle, permettant d'assurer leurs propriétés de bon fonctionnement. Cette conception induit, à la fois la spécification, le raffinement et la vérification. La spécification est la première phase du processus de conception, elle aide à capturer les exigences fonctionnelles et comportementales

du système à développer. Le raffinement permet de changer le niveau d'abstraction d'une spécification par la prise en compte de nouvelles contraintes environnementales et structuration du système à concevoir. La vérification est le processus de vérifier si le système se comporte de la manière souhaitée et répond aux propriétés attendues, elle assure entre autre l'absence des comportements indésirables. Les principaux problèmes de la vérification sont la complétude et le coût de la vérification. Les processus de spécification et de vérification d'un système sont complexes et interdépendants, par conséquent, ils sont développés de façon conjointe.

Des efforts ont été entrepris, à la fois pour offrir à la programmation distribuée temps réel des modèles formels, et pour fournir des infrastructures simples à utiliser. Ainsi les algèbres de processus, parmi lesquelles on retrouve CCS (Calulus of Communicating Systems)[Mil89] et le π-calcul[Mil93][MPW92], représentent un cadre naturel pour décrire et analyser les systèmes répartis. Elles fournissent plusieurs descriptions d'un même système à des niveaux d'abstraction différents, et des techniques permettant de montrer leur équivalence.

Le π-calcul est une extension de CCS, dans la mesure où les processus peuvent échanger des noms de canaux. Cette possibilité augmente le pouvoir expressif de ce langage, et permet de décrire les systèmes dont la topologie du réseau de communication change dynamiquement. Le prix à payer pour ce gain d'expressivité est la complexité de la vérification et d'analyse de ces systèmes.

1.2 Méthodes formelles

Les méthodes formelles sont une collection de notations et des techniques pour décrire et analyser les systèmes critiques. Ces méthodes sont dites *formelles* dans le sens où elles sont basées sur des théories mathématiques, telle que la logique, la théorie des automates et la théorie des graphes. Elles visent à améliorer la qualité de conception des systèmes.

Les techniques de spécification formelle permettent une description précise et non ambiguë des propriétés des systèmes. Les techniques d'analyse formelle peuvent être utilisées pour vérifier si un système satisfait sa spécification. Elles peuvent réduire considérablement le risque de dommages exposés par des erreurs de conception et de spécification des systèmes.

Dans les premières années de la recherche sur les méthodes formelles, l'accent a été mis sur la garantie de la correction des systèmes.

Dans le contexte des systèmes répartis de manière générale, et les systèmes critiques en particulier, la vérification formelle est l'acte de prouver ou de réfuter une propriété d'un système par rapport à une spécification formelle, en utilisant des méthodes basées sur les mathématiques, comme la logique et la théorie des graphes.

Une spécification formelle d'un système peut aider à obtenir non seulement à une meilleure description (modulaire), mais aussi à une meilleure compréhension et une vision plus abstraite du système. La vérification formelle, soutenue par des outils automatisés, peuvent détecter les erreurs dans la conception qui ne sont pas faciles à trouver en utilisant des tests, et peut être utilisée pour établir la correction de la conception. La vérification formelle est, par exemple,

été appliquée à des protocoles de communication et de cryptographie, des algorithmes distribués, des circuits combinatoires, et des logiciels exprimés en code source.

Bien que, aux premiers stades de la recherche sur les méthodes formelles, il y avait seulement une utilisation marginale des méthodes formelles dans l'industrie, de nouveaux langages, et outils sont développés au cours des années 1990 et ont stimulé un grand intérêt dans l'adaptation des méthodes formelles dans l'industrie. Les progrès dans les domaines des méthodes de spécification formelles, de la vérification basée modèle (Model Checking) [CES86] [HNSY92], et les preuves de théorèmes (Theorem Proving) ont fait promouvoir l'utilisation des méthodes formelles dans l'industrie. De ce fait, des systèmes de plus en plus complexes sont conçus selon cette démarche formelle. En effet l'essor croissant des technologies permet de définir des systèmes sophistiqués d'une complexité de plus en plus difficile à maîtriser. Ainsi, la conduite automatisée de métro, la supervision d'une centrale nucléaire, sont autant d'exemples de cette informatisation croissante des systèmes complexes.

A titre d'exemple, en 1998, le métro entièrement automatique sans conducteur a été lancé à Paris Métro et, en 2006, la navette entièrement automatique sans conducteur entre les différents terminaux de l'aéroport de Roissy est mise en service. Avec ces succès historiques s'éleva un désir dans les universités et les industries à apprendre des méthodes formelles de manière plus systématique. Aujourd'hui les méthodes formelles sont souvent emmitouflées avec des outils, disponibles en de nombreux logiciels open source, qui appuient aussi lieu la vision architecturale que la vision orientée composants des systèmes.

1.3 Systèmes répartis temps réel

Les systèmes répartis temps réel sont essentielles pour les industrielles, à savoir pour les systèmes de télécommunication modernes, les usines, les systèmes de défense, les avions, les aéroports, les stations spatiales.

En plus de leur comportement fonctionnel correct, les applications peuvent aussi avoir différents types d'exigences de temps réel. Certaines applications ont des contraintes de latence, ce qui signifie que le résultat de certains calculs doit être terminé dans un délai déterminé, appelé *deadline*. Ce type d'exigence est commun dans les applications de contrôle qui doivent réagir rapidement aux événements entrants. D'autres applications sont en pipeline et ont des exigences de débit au lieu des exigences de latence. L'important dans ce cas est le temps nécessaire pour effectuer le calcul en pipeline. Comme exemple de cette catégorie on trouve les applications de streaming temps-réel, tel qu'un décodeur vidéo qui doit être capable de présenter une image vidéo sur un écran de télévision avec une fréquence de 100Hz. Cela signifie qu'une nouvelle image doit être affichée sur l'écran toutes les 10ms. Le temps de décoder une trame peut, cependant, être supérieur à 10ms si le processus de décodage est en pipeline.

Un système temps-réel est un système informatique qui est contraint par sa spécification pour répondre non seulement aux exigences fonctionnelles, mais aussi aux exigences temporelles, appelée souvent "contraintes temporelles".

Informellement, il existe trois différences principales entre un «système temps réel» et un «système atemporel».

1. Un système temps-réel interagit étroitement avec son environnement. Un calcul activé par un stimulus de l'environnement doit être terminé avant la date limite indiquée. Si le calcul ne prend pas fin avant son échéance, ou délivre une valeur incorrecte, on considère que le système a échoué.

2. Les données temps réel peuvent être invalidées par le passage du temps.

3. Dans de nombreuses applications, il n'est pas possible d'exercer un contrôle de flux explicite sur l'environnement du système en temps réel.

Les systèmes temps réel sont généralement classés en [Kop11][Sch93] : systèmes temps réel stricts ou durs et systèmes temps réel souples ou mous. Un système temps réel dur est un système dans lequel il est impératif que les réponses se produisent dans le délai spécifié, autrement les conséquences sont graves ou potentiellement catastrophiques. Ces systèmes sont souvent critiques, tel que le domaine de l'automobile et l'aérospatiale. Un système temps-réel souple, est un système, dans lequel les contraintes temporelles peuvent ne pas être respectées dans un certain degré, sans que des conséquences graves soient produites.

1.4 Systèmes distribués

Dans la littérature, le terme *système distribué* est souvent utilisé pour exprimer des concepts différents. Plusieurs définitions émergent, parmi les quelles[TvS02] :

Définition 1.1 *Un système distribué est composé d'un ensemble de processus qui communiquent par passage de messages sur le réseau de communication.*

Définition 1.2 *Un système distribué est une collection d'ordinateurs indépendants qui apparaissent à leurs utilisateurs comme un système unique et cohérent*

Cette dernière définition a deux aspects. Le premier aspect concerne le matériel dans lequel les machines sont autonomes. Le second aspect le logiciel ; en effet le logiciel étant distribué sur les différents machines et les comportements conjoint de ces machines constituent le comportement global perçu par les utilisateurs.

Etant donné que la communication interprocessus est au cœur de tous les systèmes distribués, il devient impératif de prendre en compte les modalités de communication dans ces systèmes lors de leur étude. Ces communications sont basées sur le paradigme d'échange de messages. Les fonctionnalités sont offertes par le système de communication sous-jacent.

La distribution est définie autour de l'existence du concept de domaine[BCGL02], c'est à dire la répartition spatiale du modèle de calcul. Mais tous les modèles CCS, CSP (Communicating Sequential Processes)[Hoa78][Hoa85], ACP (Algebra of Communicating Processes)[BK84],

LOTOS (Language of Temporal Ordering Specification) [ISO88b][ISO88a], D-LOTOS (Durational LOTOS) [SC03b], π-calcul ignorent cette répartition spatiale. Cette répartition devrait être prise en compte dans la conception de ces systèmes. Par exemple, si deux machines effectuent leurs propres calculs mais échangent certaines de leurs conclusions, et que l'on essaye de modéliser cette situation en π-calcul, rien ne pourra différencier les processus représentant les calculs se déroulant sur une machine de ceux s'exécutant sur l'autre. Plusieurs formalismes sont apparus pour raisonner sur de tels systèmes répartis. En particulier le π-calcul distribué[HR02][Hen07], le Join-calcul distribué[FGJ$^+$96][Fou98], et les ambients mobiles [CG98]. Ces derniers se concentrent sur la notion de localité, les réductions de ces entités étant des réagencements de l'emboîtement de telles localités. Plus récemment, dans [JM04][Mil09] a été proposé les bigraphes, comme formalisme permettant d'unifier les ambients mobiles et le π-calcul.

1.5 Les algèbres de processus

Dans la littérature, le mot processus signifie une exécution séquentielle d'instructions[Bae04] (terminologie standard dans le domaine des systèmes d'exploitation). La définition pour la première fois d'une algèbre décrivant le comportement d'un système concurrent à fait introduire le concept d'agents qui peuvent être composés par des opérateurs de composition séquentielle, de choix, parallèle...etc. Cela conduit à la définition de l'algèbre de processus CCS. Par la suite, les mots processus et agent ont été utilisés indifféremment dans la littérature. Le mot algèbre dénote que nous prenons une approche algébrique / axiomatique pour la définition des comportements.

Une algèbre de processus se concentre sur la spécification et la manipulation des termes de processus induits par un ensemble de symboles d'opérateurs. La plupart des algèbres de processus contiennent des opérateurs de base pour construire des processus complexes. Une sémantique opérationnelle structurelle est utilisée pour donner formellement à chaque terme "processus" une représentation sémantique. Cette représentation est souvent exprimée sous forme d'un système de transitions. Le cœur des algèbres de processus est qu'ils imposent une logique équationnelle sur les termes de processus, tels que deux processus peuvent être assimilés si et seulement si leurs représentations sémantiques sont équivalentes selon une relation d'équivalence. Une algèbre de processus peut être étendue par de nouveaux opérateurs, pour améliorer son expressivité ou pour faciliter la spécification du comportement du système.

Ces concepts fondamentaux des algèbres de processus, lorsqu'ils sont combinés avec des langages appropriés pour la description des types de données, fournissent des solutions techniques, en particulier par l'expressivité offerte aux utilisateurs et le large spectre de leurs applications. Deux algèbres de processus ont été normalisées à l'ISO à savoir LOTOS et E-LOTOS (Enhanced LOTOS)[FDI01]. CCS[Mil80][Mil89], CSP, et ACP , offrent un excellent cadre pour la description des systèmes concurrents communicants, et ils sont bien équipés pour l'étude de leurs propriétés comportementales. Les logiques temporelles peuvent être

utilisées pour exprimer formellement ces propriétés.

En définissant CCS, Milner a été le premier à isoler les concepts des algèbres de processus. CCS était parmi les premiers calculs proposés pour l'étude des systèmes concurrents, et a été suivie par de nombreuses variantes. Ce calcul se compose de :

• Un langage formel simple pour décrire ces systèmes en termes de leurs structures ;

• Une théorie sémantique qui cherche à comprendre le comportement des systèmes décrits dans le langage, en termes de leur capacité à interagir avec l'environnement.

Malgré son succès, CCS ne peut décrire qu'un nombre très limité de systèmes. La restriction la plus sérieuse est que, pour tout système la topologie du réseau de connexions est statique. Cependant les systèmes interactifs modernes sont très dynamiques. Les processus de calcul, ou agents, sont très mobiles. En effet, ils se déplacent dans le réseau sous-jacent et changent dynamiquement de liens de communication. Depuis, les recherches ont été menées sur la sémantique des processus d'ordre supérieur qui permettent à des canaux de communication ou aux processus eux même d'être transportées à travers ces canaux de communications. Ces calculs de processus munis de la possibilité de créer dynamiquement et d'échanger des noms de canaux sont souvent désignés comme calculs mobiles.

Le π-calculus [MPW92][Mil93][Mil99] est une extension de CCS qui vise à traiter certains aspects dynamiques des processus. Plus précisément, il inclut la génération dynamique de canaux de communication et permet ainsi des connexions sous une topologie dynamique. Ce qui augmente considérablement sa puissance expressive.

Malgré que le π-calcul prend en charge la modélisation de la plus part des aspects comportementaux des systèmes communicants et mobiles, ce modèle reste incapable d'exprimer convenablement le lieu de déroulement des activités et leur migration. Ce qui est appelé communément le domaine d'activité.

Les recherches sur les algèbres de processus étendues avec une notion quantitative du temps ont commencé avec les travaux de Reed et Roscoe[RR86] sur une variante de CSP. Les algèbres CCS, ACP et LOTOS ont aussi fait l'objet de travaux visant à les enrichir avec des informations temporelles à savoir TCCS[MT90], ACP_ρ[BB91], ET-LOTOS[1][LL97], RT-LOTOS[2][CdS93b][CdS93a][CdO95a][CdO95b], D-LOTOS[SC03b], $tD\pi$[GP06] etc. Pour une grande partie de ces algèbres, l'outillage a été réalisé par traduction vers les automates temporisés[AD94] [AFH94], ce qui a permis de réutiliser des outils comme KRONOS[Yov97] et UPAAL [LPY97].

1.6 Mobilité

Un système distribué peut être statique, dans le sens où la topologie du réseau de connexion ne changer pas : le nombre de processus et des canaux qui les relient sont fixes, même si les états des processus changent avec le temps. L'inverse de ceci est un système dynamique.

[1] Enhanced Timed-LOTOS

[2] Real Time LOTOS

Malgré toutes les propriétés qui peuvent déjà être étudiées dans le cadre de CCS, ce calcul impose une structure de communication figée : l'ensemble des noms partagés par deux processus ne peut pas varier au cours de l'exécution. L'ensemble des comportements que le calcul peut modéliser est donc relativement restreint. Le π-calcul [Mil93][Mil99], introduit par Robin Milner, Joachim Parrow et David Walker dans [MPW92], résout cette difficulté et franchit une grande étape en permettant de créer de nouveaux canaux et de communiquer des noms de canaux sur un canal. La mobilité est modélisée par la manière dont les processus utilisent les noms des canaux, les noms des canaux peuvent s'échanger entre les processus. Par exemple, un processus P pourrait créer un nouveau nom de canal, et de le transmettre aux processus Q et R. L'aspect le plus intéressant dans le modèle π-calcul provient de la capacité des processus à créer de nouveaux noms de canaux et de les transmettre à d'autres processus, donc la topologie du réseau de connexion change de manière dynamique. Ce saut conceptuel aboutit alors à un calcul de processus dans lequel de nouveaux liens entre processus peuvent se créer et se détruire dynamiquement.

Quelles sont les entités mobiles et dans quel espace se déplacent-t-elle ? la mobilité est classée en trois catégories principales[Mil99] :

(A) Un processus se déplace, dans l'espace physique de calcul des sites ;

(B) Un processus se déplace, dans l'espace virtuel des processus liés ;

(C) Liens qui se déplacent, dans l'espace virtuel des processus liés.

Les systèmes mobiles sont des systèmes de processus dont la topologie des communications peut varier au cours du temps. La possibilité de changer des liens de communications peut être interprétée comme le mouvement d'un processus dans un espace virtuel. Dans les langages qui permettent de décrire de tels systèmes, la mobilité est modélisée par un mécanisme de génération dynamique de noms, plus la possibilité d'insérer des noms des canaux dans les messages échangés.

Traditionnellement, la migration de code dans les systèmes distribués prend la forme de migration de processus dans laquelle tout un processus se déplace d'une machine à une autre. Déplacement d'un processus en cours d'exécution à une machine différente est une tâche coûteuse et complexe. En général cette migration est réalisée pour améliorer les performances. L'idée de base est que la performance globale du système peut être améliorée si les processus sont déplacés à partir des machines lourdement chargées à des machines légèrement chargées.

Prise en charge de la migration du code peut également aider à améliorer les performances en exploitant le parallélisme, mais sans les subtilités habituelles liées à la programmation parallèle. Un exemple typique est la recherche d'informations dans le Web. Il est relativement simple à mettre en œuvre une requête de recherche sous la forme d'un petit programme portable qui se déplace de site en site. En faisant plusieurs copies d'un tel programme, et l'envoi de chaque copie aux différents sites, nous pourrons retourner l"information requise avec une vitesse linéaire par rapport au nombre d'instances.

Outre l'amélioration de performances, il y a d'autres raisons pour supporter la migration de code. La plus important est celle de la flexibilité. Si le code peut se déplacer entre les

différentes machines, il devient possible de configurer dynamiquement les systèmes distribués.

Le modèle de migration le plus simple est la mobilité faible. Dans ce modèle, il est possible de transférer uniquement le segment de code, avec éventuellement quelques données d'initialisation. Une des caractéristiques de la mobilité faible est qu'un processus déplacé, démarre toujours son exécution à partir de son état initial. C'est ce qui arrive, avec les applets Java. L'avantage de cette approche est sa simplicité. La mobilité faible exige seulement que la machine cible soit capable d'exécuter ce code, c'est à dire sa portabilité.

Contrairement à la mobilité faible, dans des systèmes qui supportent la mobilité forte, le segment d'exécution peut être transféré. L'avantage réside dans le fait qu'un processus en cours peut être arrêté, ensuite déplacé vers une autre machine, puis reprendre son exécution là où il s'est arrêté. Néanmoins, la mobilité forte est plus difficile à mettre en œuvre en comparaison avec la mobilité faible.

1.7 Critères de classification des modèles distribués et mobiles

Cette section a pour objectif d'analyser, les critères permettant plus ou moins une classification des modèles de la distribution. Pour cela, nous nous appuyons en grande partie sur les documents réalisé dans le cadre du projet MARVEL [BCGL02][BGL00].

1.8 Les critères de la distribution

L'analyse de la distribution sera développée autour de la notion de **domaine (site, localité)**. Plus précisément, la distribution sera définie comme l'existence de domaines. Un domaine peut être défini comme étant une région regroupant un ensemble de processus, et ce de manière implicite ou explicite. Dans la suite, nous employons aussi les termes « site » et « localité » avec le même sens que « domaine ». Un domaine peut donc être désigné, en tant que cible d'une communication distante par exemple, ou encore en tant que cible d'une migration de processus ou d'une migration de domaine. À la notion de domaine, est souvent associée celle de migration permettant la mobilité d'entités entre les différents domaines d'un système. Sans la migration, les modèles de la distribution semblent présenter peu d'intérêt. D'autre part, les langages modernes permettant la programmation d'applications réparties implantent des primitives de migration et ainsi accroissent la variété des applications réalisables. L'analyse de la distribution se limite donc à envisager les modèles selon deux aspects : la répartition et la mobilité. D'autres dimensions de la distribution pourraient être prises en compte dont par exemple la sécurité et l'implantation des langages dédiés.

1.8.1 Aspect répartition

Les critères décrits ici sont : la topologie de l'espace des domaines, les sémantiques de domaine, les capacités d'observation/contrôle des domaines et enfin les caractéristiques de com-

munications intra- et inter-domaines. La défaillance est également abordée comme critère de comparaison.

Topologie de l'espace des domaines

Il s'agit ici de décrire la façon dont les domaines sont structurés. Dans la plupart des cas, la structure est hiérarchique : un domaine peut contenir plusieurs sous-domaines. Un système est alors un arbre de domaines ou encore une forêt. Le fait de structurer les domaines sous la forme d'une arborescence permet d'expliciter assez naturellement la migration des domaines les uns par rapport aux autres, sous la forme d'une reconfiguration locale de l'arborescence. Dans une arborescence de domaines, chaque nœud désigne un domaine, abritant ses propres processus et possédant un certain nombre de domaines fils. Certains des calculs sous-tendent l'existence d'un éther entre les domaines, dans lequel les messages sont acheminés d'une localité à une autre.

L'autre type de structure est en fait un cas particulier du précédent : il s'agit des espaces de domaines plats.

Sémantiques de domaines

Les domaines permettent d'unifier le comportement des entités qu'ils regroupent suivant diverses orientations.

- Orientation vers la communication : On parle de communication intra-domaine ou locale. La communication entre des entités appartenant à des domaines différents est dans ce cas impossible.
- Orientation vers la mobilité : comporte une primitive atomique de migration du contenu d'une localité. Dans le cas d'une structure hiérarchique, ce contenu peut être un sous-domaine emporté avec tous ses domaines fils.
- Orientation défaillance : se traduit par l'incapacité à émettre et/ou à recevoir des messages ou des entités migrantes.

Observabilité / Contrôlabilité des domaines

Il s'agit de l'aptitude du calcul à permettre l'observation et/ou le contrôle des domaines depuis l'application. Si cette notion est généralement liée à celle de défaillance, le contrôle peut être motivé par des considérations de sécurité. Ainsi, le contrôle peut, par exemple, explicitement permettre l'accès à un domaine, ou encore l'activation d'un domaine.

Domaines et communication locale

On regroupe ici les caractéristiques de la désignation locale, de la politique de communication locale et de la sémantique de la réception. La désignation d'une ressource (généralement un

récepteur) se fait grâce à un nom éventuellement connu à l'issue d'une communication. Dans un domaine, un même nom peut représenter un ou plusieurs récepteurs. Il est parfois possible de communiquer des usages spécifiques appelés « capacités ». La réception peut être unique ou multiple (plusieurs récepteurs identifiés par un même nom). Elle est souvent statique dans le sens où la réception d'un nom de canal n'alloue que la capacité d'émettre sur celui-ci.

Domaines et communication distante

La référence a une ressource distante par son nom est obtenue grâce à la notion de portée distribuée, c'est-à-dire que la portée d'un nom peut s'étendre à plusieurs domaines. La communication distante peut être globale, restreinte à un voisinage ou purement locale comme dans le cas précédent. La localité cible de la communication peut être explicite ou implicite (ce cas accompagne souvent celui de la communication globale ou restreinte). Le transport du message est soit direct (ou atomique) soit indirect (ou en plusieurs phases où il doit d'abord sortir du domaine source, pénétrer dans le domaine cible et enfin être consommé). Le routage « logique » est l'acheminement dans l'arborescence des domaines. Il peut être explicite ou transparent.

Domaines et défaillances

Certains modèles fondent leur sémantique sur le concept de défaillance. La défaillance d'une localité signifie son incapacité à émettre/recevoir des messages ou des entités migrantes. C'est une sorte de blocage temporaire ou définitif d'un domaine et de son contenu. Généralement, la défaillance est contrôlable et observable depuis l'extérieur ou l'intérieur d'une localité.

1.8.2 Aspect mobilité

L'analyse de la mobilité s'articule autour de trois notions :

– Les domaines : Il s'agit de déterminer ici l'objet de la migration et le domaine d'origine et cible.

– Les entités migrantes : Ce sont les objets migrants. Ces objets peuvent comporter des contenus exécutables.

– Le contexte : C'est l'environnement d'une entité au moment de sa migration.

Ces notions ne sont pas exclusives ; elles sont interdépendantes suivant les choix optés pour chacune d'elles. Une entité migrante est toujours incluse dans un domaine, elle peut même être un domaine. Parfois, le contexte est inclus dans l'entité mobile. Cependant, le contexte s'étend généralement au-delà de l'entité voir de son domaine initial et peut donc être partagé avec d'autres entités. Il convient alors de déterminer la politique à adopter pour le contexte vis-à-vis de l'entité mobile.

Une première distinction que nous pouvons faire sur les entités mobiles concerne leur caractère statique ou dynamique. En effet, si elles contiennent du code exécutable, celui-ci

peut être soit activé après migration (dans ce cas on dit que l'entité est statique), soit indépendant de la migration, c'est-à-dire que son exécution est interrompue durant la migration. Dans ce dernier cas, l'entité est dynamique et on peut encore distinguer deux cas : celui où l'instruction de migration est bloquante vis-à-vis de l'entité migrante ou asynchrone (elle peut entrer en concurrence avec des instructions de migration). La mobilité peut s'opérer par déplacement ou par réplication. Dans le premier cas, l'entité mobile est un code original, dans le second, c'est une copie (ou « clone ») d'une portion de code dont l'original reste en place. En pratique, cette distinction concerne essentiellement le contexte ; pour les entités migrantes les deux types de mobilité sont généralement modélisés (le plus souvent c'est la réplication qui peut être simulée). On peut également différencier deux points de vues vis-à-vis de l'autorité (ou encore l'instruction) qui commande la migration. L'autorité peut être extérieure à l'entité mobile ; on parle alors de mouvement objectif. Dans le cas contraire, c'est-à-dire si la migration est directement commandée par l'entité objet du mouvement, alors on parle de mouvement subjectif. Les entités mobiles statiques font toujours l'objet de mouvements objectifs. Le comportement du contexte « public » vis-à-vis d'une entité en cours de migration est généralement celui de la liaison dynamique : la valeur d'un nom public est celle qu'il a dans le site d'exécution courant. En ce qui concerne la partie « privée » du contexte, celle-ci pouvant être partagée avec des entités fixes, plusieurs comportements sont alors envisageables :

– Contexte fixe : Les éléments du contexte restent dans le domaine initial. Dans ce cas la relation est soit rompue (avec éventuellement liaison dynamique sur le site d'arrivé), soit est maintenue à distance implicitement ou explicitement (par l'intermédiaire de proxy par exemple).

– Contexte mobile : On retrouve ici les types de mobilité par déplacement et par réplication. Le premier cas nous fait revenir dans un problème symétrique au précédent vis-à-vis des entités fixes. Dans le second, il faut déterminer si la cohérence entre les éléments du contexte original et de sa copie doit être maintenue.

1.9 Contributions

Notre travail s'inscrit dans le cadre de la spécification des systèmes distribués, temps réel, et mobiles. Dans ce contexte notre principale contribution est la définition d'un modèle qui intègre trois concepts largement utilisés dans les systèmes concurrents : la distribution, les contraintes temporelles, et la mobilité.

Distribution & communication **:** La distribution implique le concept de domaine ou de localité. Les calculs de processus traitent cette caractéristique d'une façon implicite ou explicite. Le calcul des ambient[Car97] [CG98] propose des primitives de domaines explicites, appelés ambients. Le Join-calcul distribué[FGJ+96][Fou98] propose une autre manière de traitement tel que la solution chimique locale et la solution distribuée. Dans notre approche nous avons proposé un langage distribué communiquant (DD-LOTOS), et un modèle

sémantique distribué communiquant (C-DATA). Le langage DD-LOTOS permet de spécifier les systèmes communicants. Cette spécification est traduite de manière opérationnelle vers le modèle sémantique C-DATA, pour une éventuelle vérification formelle.

Contraintes temporelles **:** Les algèbres de processus ET-LOTOS[3][LL97], RT-LOTOS[4] [CdS93b][CdS93a][CdO95a][CdO95b], etc, intègrent de manière explicite des opérateurs pour exprimer les contraintes temporelles. Ces algèbres sont définies sous une sémantique d'entrelacement, impliquant l'atomicité structurelle et temporelle des actions. Le langage D-LOTOS[SC03b], est défini sur un autre modèle sémantique dit de vrai parallélisme à la place de la sémantique classique d'entrelacement. Il intègre à la fois les contraintes temporelles et les durées d'actions. Les contraintes temporelles se traduisent en générale par des exigences sur la façon d'observer et/ou de piloter l'environnement. Nous avons adopté dans notre approche les modèles définis sur les sémantiques de vrai parallélisme. De ce fait, nos modèles prennent en charge les contraintes temporelles exprimées grâce aux opérateurs définis dans le langage, et les durées d'actions grâce à la sémantique de maximalité.

Mobilité **:** Les modèles les plus anciens pour les systèmes distribués sont statiques. Ainsi, les modèles étaient moins complexes et permettaient de traiter certains problèmes dans un cadre plus simple. Pourtant, la mobilité est classique dans les systèmes d'exploitation. Parmi les formalismes présentés qui prennent en compte la mobilité, on trouve le $\pi-$calcul. La mobilité se traduit par l'existence de noms de canaux, qui sont inséparables du processus de communication. L'existence de noms suggère de plus un espace abstrait de processus connectés, dans lequel les noms représentent les connexions. Seuls les processus qui partagent des noms sont alors en mesure d'interagir. La structure d'un système change donc d'une manière dynamique, car les liens entre processus sont sans cesse crées et détruits.

Pour prendre en charge cet aspect nous avons défini un langage qui permet la programmation mobile (Mobile DD-LOTOS), et un modèle sémantique mobile C-DATA. Dans notre approche la mobilité s'exprime par la migration des processus entre les différentes localités, la création de nouvelles localités et enfin la suppression des ces dernières.

1.10 Plan du document

Ce document est organisé en deux parties, de la manière suivante :

La première partie introduit les concepts liés à notre travail. Elle est composée des chapitres suivants :

Chapitre 1 : **Introduction générale**. Ce chapitre introduit le contexte général de notre étude, des notions préliminaires sur les méthodes formelles, les systèmes distribués et temps réel, les algèbres de processus, et la mobilité. Et nous discutons les critères d'analyse des systèmes concurrents à savoir la distribution et la mobilité.

Chapitre 2 : **Modèles et calculs des processus distribués et mobiles**. Ce chapitre

[3]Enhanced Timed-LOTOS
[4]Real Time LOTOS

expose en détaille les modèles algébriques de spécification des systèmes concurrents, à savoir CCS, LOTOS, $\pi-$calcul, join-calcul, calcul des ambients...

Chapitre 3 : **Modèles et calculs temps réel**. Ce chapitre introduit les modèles algébriques de spécification des systèmes temps-réel tels que RT-LOTOS et D-LOTOS. On termine par une discussion sur les limites de D-LOTOS.

La deuxième partie comporte nos contributions. Cette partie est composée des chapitres suivants :

Chapitre 4 : **Modèle sémantique distribué pour les systèmes temps-réel distribués et mobiles**. Dans ce chapitre nous définissons deux modèles sémantiques à savoir C-DATA et Mobile C-DATA, le premier prend en charge les aspects de distribution et de communication. Alors que le deuxième prend l'aspect de mobilité. Ensuite un algorithme de traduction des DATA* vers les automates temporisés a été proposé.

Chapitre 5 : **Distributed D-LOTOS : Un modèle de spécification des systèmes distribués**. Ce chapitre définit un modèle permettant de spécifier des systèmes temps réel et distribués communicants.

Chapitre 6 : **Mobile DD-LOTOS**. Ce chapitre définit un modèle permettant de spécifier des systèmes temps réel, distribués et mobiles.

Chapitre 7 : **Conclusion et perspectives**. Dans ce chapitre nous concluons et présentons les ouvertures possibles.

Première partie

Modèles et Calculs

Chapitre 2

Modèles et calculs des processus distribués et mobiles

Sommaire

Ce chapitre introduit différents modèles algébriques de description formelle pour la spécification des systèmes concurrents et mobiles. Pour une analyse comparative plus détaillée de ces modèles, le lecteur pourra par exemple se reporter à [BGL00] [BCGL02] [Cas01].

La première section expose le formalisme CCS et LOTOS en termes de syntaxe et sémantique opérationnelle. La seconde expose le modèle $\pi-$calcul, extension de CCS avec mobilité, et un ensemble de variantes de π-calcul. La troisième expose le Join-calcul : un calcul pour la programmation répartie et mobile, ce modèle s'inspire des deux modèles du parallélisme $\pi-$calcul d'une part et la machine chimique abstraite de Berry et Boudol d'autre part[BB92]. La quatrième section expose le calcul des ambients, qui s'inspire du $\pi-$calcul. La cinquième section présente les langages KLAIM, M-Calcul, Obliq, Pict & NomadicPict. D'autres formalismes basés sur les réseaux de Petri pour la spécification des systèmes mobiles, ont été également introduits[KC08a][KC08b][KC10]. Finalement on termine par une conclusion.

2.1 Introduction aux algèbres de processus

Le comportement d'un système est généralement constitué de processus et de données. Les processus sont les mécanismes de contrôle pour la manipulation des données. Alors que les processus sont dynamiques et actifs, les données sont statiques et passives. Le comportement d'un système tend a être composé de plusieurs processus qui sont exécutés simultanément, ces processus s'échangent de données afin d'influencer le comportement du système.

2.1.1 CCS : Calculus of Communicating Systems

Le formalisme CCS[Mil80][Mil83] décrit les systèmes communicants comme des ensembles d'automates non-déterministes (appelés agents ou processus) qui interagissent par le biais de synchronisations. Le comportement d'un agent n'est décrit que partiellement par l'ensemble de ses traces (une trace est une suite de transitions) et la notion de bisimulation propre à cette théorie permet de raffiner la notion d'égalité des agents.

Les actions constituent l'alphabet de base des processus. On suppose un ensemble dénombrable d'actions a, b, c ... chacune ayant un inverse unique $\overline{a}, \overline{b}, \overline{c}$... . Il existe aussi une action particulière τ(tau) considérée comme interne et silencieuse. Le sens intuitif de la notion d'inverse est que l'action et son inverse constituent des actions pouvant se synchroniser si elles proviennent de deux processus communicants.

La syntaxe des processus CCS est la suivante.

$$\begin{array}{ll} \alpha & ::= a(x) \mid \overline{a}.v \mid \tau \\ P, Q, R & ::= \mathbf{0} \mid a.P \mid P + Q \mid (P|Q) \mid (v\alpha)P \end{array}$$

FIG. 2.1 – *Syntaxe de CCS*

Dans CCS on a trois types d'actions :

$a(x)$: sur le canal a on peut recevoir n'importe qu'elle valeur.

$\overline{a}.v$: émettre la valeur v sur le canal a.

τ : action interne ou silencieuse.

Un processus P peut être soit le processus inerte $\mathbf{0}$, soit un processus préfixé par une action a, soit une composition parallèle de processus $P|Q$, ou le processus $(v\alpha)P$ de restriction, c'est à dire seul le processus P a accès sur le nom α.

La signification de la syntaxe des processus est donnée par une relation de transition $P \xrightarrow{a} Q$, qui signifie "le processus P peut effectuer l'action a et devenir le processus Q". Cette relation est définie par les règles ci-dessous (Figure : 2.2).

$$\text{Act}\ \ \alpha.P \xrightarrow{\alpha} P' \qquad \text{Sum1}\ \ \frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'} \qquad \text{Sum2}\ \ \frac{Q \xrightarrow{\alpha} Q'}{P + Q \xrightarrow{\alpha} Q'}$$

$$\text{Par1}\ \ \frac{P \xrightarrow{\alpha} P'}{P \mid Q \xrightarrow{\alpha} P' \mid Q} \qquad \text{Par2}\ \ \frac{Q \xrightarrow{\alpha} Q'}{P \mid Q \xrightarrow{\alpha} P \mid Q'}$$

$$\text{Com1}\ \ \frac{P \xrightarrow{a(x)} P'Q \xrightarrow{\overline{a}v} Q'}{P \mid Q \xrightarrow{\tau} P'\{v/x\} \mid Q} \qquad \text{Com1}\ \ \frac{P \xrightarrow{\overline{a}v} P'Q \xrightarrow{a(x)} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q\{v/x\}}$$

$$\text{Res}\ \ \frac{P \xrightarrow{\alpha} P' \alpha \notin \{a, \overline{a}\}}{P + Q \xrightarrow{\alpha} P'}$$

FIG. 2.2 – *Sémantique opérationnelle de CCS*

La sémantique opérationnelle, détermine donc pour tout processus ses comportements ou traces possibles comme s'il s'agissait simplement d'une notation particulière pour un automate non-déterministe dont tous les états atteignables seraient acceptants. Mais à la différence d'un automate fini classique, les comportements d'un processus communiquant doivent tenir compte des traces infinies c'est-à-dire ne pas considérer tout comportement infini comme une "erreur" à éviter. Au contraire, ils représentent souvent des programmes serveurs

dont le comportement est volontairement et naturellement infini, puisque c'est alors l'arrêt qui peut-être considéré comme une erreur. Le langage de traces d'un processus communiquant est donc en général un ensemble de mots finis et infinis sur l'alphabet des événements.

Mais en plus d'engendrer ses propres comportements de manière autonome, un processus est aussi caractérisé sur sa "connectique" c'est à dire son interaction avec l'environnement. Par exemple $(a.P|\overline{a}.P)$ pouvant émettre a, il peut aussi se synchroniser avec un processus de la forme $\overline{a}.Q$. Or cet aspect du comportement n'est pas décrit par le langage de traces qui "oublie" la structure exacte des transitions possibles. Milner propose une relation d'équivalence entre processus qui soit plus fine que la simple équivalence de traces (égalité des langages de traces). C'est la bisimulation forte en CCS que nous allons expliquer par un exemple avant de la définir formellement.

Soient les processus $A = a.(b.P + c.Q)$ et $B = a.b.P + a.c.Q$. On constate que leurs traces sont les mêmes : trace vide, a, ab, ac, ab suivie d'une trace de P ou ac suivie d'une trace de Q. Mais les comportements de A et B diffèrent de la manière suivante. Le processus obtenu après $A \xrightarrow{a}$ peut réaliser l'une ou l'autre des actions $\{b, c\}$, il peut par exemple se synchroniser indifféremment avec $\overline{b}.R$ ou $\overline{c}.R$. Ce n'est pas le cas du processus obtenu après $B \xrightarrow{a}$ puisque celui-ci n'est pas uniquement déterminé. S'il s'agit de $b.P$, il "bloque" l'action c, c'est-à-dire qu'il ne pourrait se synchroniser avec $\overline{c}.R$ et s'il s'agit de $\overline{c}.Q$ il bloque l'action b. Cette différence de comportement est bien visible sur les systèmes de transitions de A et de B (Voir la figure 2.3) :

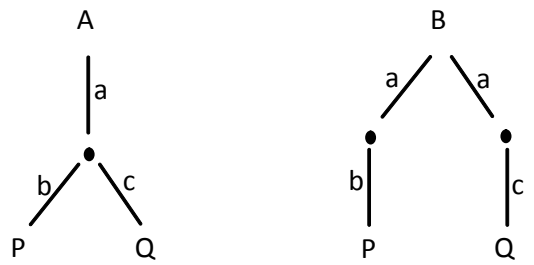

Fig. 2.3 – Comportement 1

Alors que l'ensemble des traces, étant justement un ensemble, n'encode pas les deux occurrences de $B \xrightarrow{a}$ menant à des comportements différents pour B.

Autre exemple de la nécessité d'une notion de bisimulation. Soit **0** un processus qui ne peut effectuer aucune transition, alors du point de vue de la bisimulation on veut aussi distinguer des processus comme dans la figure 2.4.

Car la trace initiale $\xrightarrow{a}$ peut, pour le processus de droite, mener à une situation de blocage ce qui est faux pour le processus de gauche. Or les traces des deux processus sont les mêmes : trace vide, a, ab,ou ab suivie d'une trace de P.

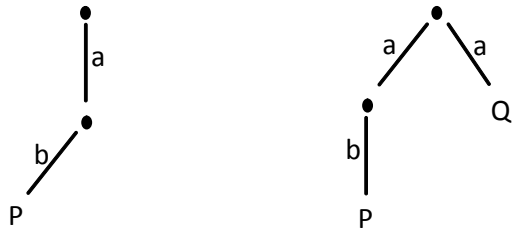

FIG. 2.4 – Comportement 2

On pourrait penser que la bonne notion de bisimulation est l'équivalence des graphes de transition ce qui reviendrait à dire que deux processus seraient bisimilaires si et seulement si leurs graphes de transitions sont égaux. Mais ce serait alors une relation trop fine car certains graphes de transition ont des symétries et ne sont que le dépliement d'un autre graphe, par exemple dans la figure 2.5, ne diffèrent, ni par leurs traces ni par leurs comportements possibles ; aucun processus extérieur saurait détecter de différence entre ces deux processus. Il sera donc normal de les identifier par la relation de bisimulation. La définition exacte de la bisimulation forte est formulée récursivement le long des transitions.

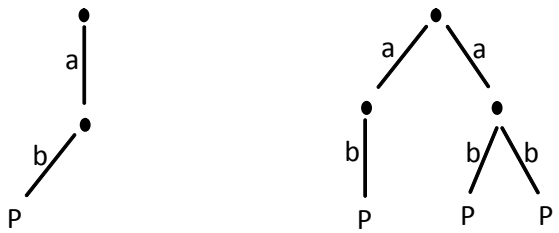

FIG. 2.5 – Comportement 3

Définition 2.1 *: Une relation binaire $\backsim$ entre processus CCS est une bisimulation forte si, $P \backsim Q$ implique que :*

1. Toute transition $P \xrightarrow{a} P\prime$ implique l'existence d'une transition $Q \xrightarrow{a} Q\prime$ telle que $P\prime \backsim Q\prime$

2. Et vice-versa : toute transition $Q \xrightarrow{a} Q\prime$ implique l'existence d'une transition $P \xrightarrow{a} P\prime$ telle que $P\prime \backsim Q\prime$.

On dit que deux processus P et Q sont bisimilaires s'il existe une bisimulation forte $\backsim$ pour laquelle $P \backsim Q$ et on écrit alors "$P \backsim Q$".

Proposition 2.1 *Pour toute bisimulation forte $\backsim$ et les processus P, Q, R on a :*

$P + Q \backsim Q + P$
$P + (Q + R) \backsim (P + Q) + R$
$P + P \backsim P$

2.1.2 Algèbre de processus à synchronisation multiple : LOTOS

LOTOS[ISO88b][ISO88a][Loh02], est une technique de description formelle, promue au rang de norme ISO en 1988.

Il s'appuie sur le langage CCS de Milner (étendu par un mécanisme de synchronisation multiple hérité de CSP [Hoa85] de Hoare) pour la spécification de la partie comportementale ; la partie description des structures de données est inspirée de Act-One[EM85], un formalisme de description des types de données abstraits algébriques.

Le concept sous-jacent à LOTOS est que tout système peut être spécifié en exprimant les relations qui existent entre les interactions constituant le comportement observable des composantes du système. En LOTOS, un système est vu comme un processus, qui peut être constitué de sous-processus, un sous-processus étant un processus en lui-même. Une spécification LOTOS décrit ainsi un système par hiérarchie de processus. Un processus représente une entité capable de réaliser des actions internes (non-observable) et d'interagir avec d'autres processus qui forment son environnement.

Les définitions de processus sont exprimées par la spécification d'expressions de comportement qui sont construites à partir d'un ensemble réduit d'opérateurs donnant la possibilité d'exprimer des comportements aussi complexes que l'on désire. Les processus sont en général définis récursivement, et le rendez-vous multidirectionnel constitue le mécanisme de base pour la communication interprocessus. Parmi les opérateurs, ceux de préfixage (;), de choix non-déterministe ([]), de composition parallèle (|[.]|) et d'intériorisation (hide) jouent un rôle fondamental.

Présentation de Basic LOTOS

Basic-LOTOS est un sous ensemble de LOTOS où les processus interagissent entre eux par synchronisation pure, sans échange de valeurs. En Basic-LOTOS les actions sont identiques aux portes de synchronisation des processus. Les principaux opérateurs de Basic-LOTOS sont listés dans le tableau 2.1 :

Syntaxe de Basic- LOTOS

- Soit P l'ensemble des identifiants de processus.
  - Soit $X \in P$.
  - Soit $\mathcal{G}$ l'ensemble des portes définissables (les actions observables en Basic-LOTOS).
  - Soient $g, g_1 ... g_n \in \mathcal{G}$.
  - Soit L un sous-ensemble (pouvant être vide) quelconque de $\mathcal{G}$ noté $L = g_1 ... g_n$.
  - Soit i l'action interne.

Opérateur		Notation	Description informelle
Inaction		`stop`	Processus de base n'interagissant pas avec son environnement
Terminaison avec succès		`exit`	Processus qui se termine (action δ) et se transforme en `stop`
Préfixage par une action	non observable	$i; P$	Processus qui réalise l'action i ou g, puis se transforme en P
	observable	$g; P$	
Choix non-déterministe		$P_1 [] P_2$	Processus qui se transforme en P_1 ou en P_2 suivant l'environnement
Composition parallèle	cas général	$P_1 \|[g_1, ...g_n]\| P_2$	P_1 et P_2 s'exécutent en parallèle et se synchronisent sur les portes $g_1, ...g_n$ et δ
	asynchrone	$P_1 \|\|\| P_2$	P_1 et P_2 s'exécutent en parallèle sans se synchroniser (sauf sur δ)
	synchrone	$P_1 \|\| P_2$	P_1 et P_2 s'exécutent en parallèle et se synchronisent sur chaque porte visible
Intériorisation		*hide* $g_1, ...g_n$ *in* P	Les actions $g_1, ..., g_n$ sont cachées à l'environnement de P et deviennent des actions internes
Composition séquentielle		$P_1 \gg P_2$	P_2 est activé dès que P_1 se termine.
Préemption (interruption)		$P_1 [> P$	P_2 peut interrompre P_1 tant que P_1 ne s'est pas terminé

TAB. 2.1 – *Principaux opérateurs de Basic-LOTOS*

La syntaxe formelle du Basic-LOTOS est donnée par la figure 2.6

Processus $X[g_1, ..., g_n] := P$ **endproc**

$$\begin{array}{rcl} P & ::= & \textbf{stop} \mid \textbf{exit} \mid X[L] \mid i;P \mid g;P \\ & \mid & P[]P \mid P|[L]|P \mid \textbf{hide } L \textbf{ in } P \\ & \mid & P >> P \mid P[> P \end{array}$$

FIG. 2.6 – *Syntaxe de Basic-LOTOS*

Sémantique opérationnelle de Basic-LOTOS

La tableau 2.2, page 29 présente les règles d'inférences de la sémantique opérationnelle de Basic-LOTOS. Les notations suivantes seront utilisées :

- δ est l'action de terminaison de Basic-LOTOS.
- $\mathcal{G}^i = \mathcal{G} \cup \{i\}, \mathcal{G}^\delta = \mathcal{G} \cup \{\delta\}, \mathcal{G}^{i,\delta} = \mathcal{G} \cup \{i, \delta\}$
- $P \xrightarrow{g} P'$, signifie que le processus P peut réaliser l'action g et se comporte ensuite comme $P\prime$.

Présentation de full LOTOS

Full LOTOS (ou simplement LOTOS), est Basic-LOTOS étendu avec la possibilité d'échanger des valeurs lors des synchronisations entre processus. Contrairement à Basic-LOTOS, en full LOTOS une action n'est pas simplement une porte de synchronisation, mais une porte plus des données échangées lors de la synchronisation.

Les structures de données et les opérations associées sont définies par le langage Act-One qui formalise la spécification de types abstraits de données : il définit les propriétés essentielles des données et les opérations qu'une implémentation correcte du type doit assurer.

Les valeurs en LOTOS peuvent être :

- Echangées entre processus lors d'une synchronisation,
- Employées dans les prédicats de garde des processus,
- Utilisées comme paramètres pour la définition des processus et pour l'instanciation de valeurs,
- Associées à l'opérateur de choix généralisé,
- Exportées lors d'une terminaison avec succès d'un processus,

Avec l'ajout des données dans la spécification, les actions deviennent des entités qui adjoignent à une porte trois composantes de base, comme cela est illustré dans le tableau 2.3 page 30. Une action peut :

- Offrir une valeur $x(!x)$,
- Accepter une valeur $y(?y : type)$,
- Incorporer des prédicats qui conditionnent l'acceptation de valeurs.

$exit \xrightarrow{\delta} stop$
$g; P \xrightarrow{g} P \quad (g \in \mathcal{G})$
$i; P \xrightarrow{i} P$
$\dfrac{P \xrightarrow{g} P'}{P[]Q \xrightarrow{g} P'} \quad (g \in \mathcal{G}^{i,\delta})$
$\dfrac{P \xrightarrow{g} P' \quad Q \xrightarrow{g} Q'}{P\|[L]\|Q \xrightarrow{g} P'\|[L]\|Q'} \quad (g \in L \cup \{\delta\})$ $\dfrac{P \xrightarrow{g} P'}{P\|[L]\|Q \xrightarrow{g} P'\|[L]\|Q} \quad (g \in \mathcal{G}^{i} \diagdown \{\delta\})$
$\dfrac{P \xrightarrow{g} P' \quad (g \in \mathcal{G}^{i,\delta} \diagdown L)}{hide\ L\ in\ P \xrightarrow{g} hide\ L\ in\ P'}$ $\dfrac{P \xrightarrow{g} P' \quad (g \in L)}{hide\ L\ in\ P \xrightarrow{i} hide\ L\ in\ P'}$
$\dfrac{P \xrightarrow{g} P'}{P \gg Q \xrightarrow{g} P' \gg Q} \quad (g \in \mathcal{G}^{i})$ $\dfrac{P \xrightarrow{\delta} P'}{P \gg Q \xrightarrow{i} Q}$
$\dfrac{P \xrightarrow{g} P'}{P[> Q \xrightarrow{g} P'[> Q} \quad (g \in \mathcal{G}^{i})$ $\dfrac{Q \xrightarrow{g} Q'}{P[> Q \xrightarrow{g} Q'} \quad (g \in \mathcal{G}^{i,\delta})$ $\dfrac{P \xrightarrow{\delta} P'}{P[> Q \xrightarrow{\delta} Q}$
$\dfrac{P_X[g_1/g'_1 ... g_n/g'_n] \xrightarrow{g} Q' \quad X[g_1 ... g_n] = P_X}{X[g_1 ... g_n] \xrightarrow{g} Q'} \quad (g \in \mathcal{G}^{i,\delta})$ $\dfrac{P \xrightarrow{g} P'}{P_\phi \xrightarrow{\phi\{g\}} P'_\phi} \quad (\phi = [g_1/g'_1 ... g_n/g'_n])$ $(g \in \mathcal{G}^{i,\delta})$

TAB. 2.2 – *Sémantique opérationnelle de Basic-LOTOS*

$\mathbf{P_1}$	$\mathbf{P_2}$	**Condition de synchronisation**	**Type d'interaction**	**Résultat**
$g!v_1$	$g!v_2$	$valeur(v_1) = valeur(v_2)$	Concordance des valeurs	Synchronisation
$g!v$	$g?x : T$	$v \in T$	Passage de valeur	Après synchronisation, $x = valeur(v)$
$g?x : T$	$g?y : U$	$T = U$	Choix aléatoire d'une valeur	Après synchronisation, $x = y = v$ avec $v \in T$(ou $v \in U$)

TAB. 2.3 – *Règles de synchronisation en LOTOS avec données*

Le tableau 2.3 illustre les nouvelles règles en prenant l'exemple de deux processus P_1 et P_2 composés par une synchronisation sur la porte g (la fonction *valeur* renvoie la valeur d'une variable). La règle veut qu'après une synchronisation, les actions impliquées dans cette synchronisation doivent offrir des variables de même types et de même valeurs.

La déclaration de processus est étendue avec l'usage de paramètres. Par exemple :

$$\textbf{Processus} \quad X[g_1, ...g_n](x_1 : T_1, x_m : T_m) := P \quad \textbf{endproc}$$

Définit, pour le processus X, m variables $x_1...x_m : T_1...T_m$ qui le paramètrent. Ainsi, ce processus devra être instancié avec une liste de valeurs $v_1...v_m$ affectées à ces paramètres : $X[g_1, ...g_n](v_1...v_m)$

Le tableau 2.4, page 31 liste les principaux opérateurs introduits dans LOTOS pour manipuler les données. Dans ce tableau, x correspond à un nom de variable, $Type$ à un type de données et *pred* à une expression logique (prédicat de garde).

Sémantique opérationnelle de LOTOS

Le tableau 2.5, page 31 présente de quelle manière la sémantique de Basic LOTOS est enrichie pour donner la sémantique opérationnelle de LOTOS.

2.2 π-calcul

Extension de CCS, π-calcul[MPW92][Mil93][Mil99] est un calcul de processus parallèles, qui permet de rendre compte de systèmes dont la topologie de communication change dynamiquement. Ce calcul est maintenant considéré comme le fondement de la programmation concurrente[SW01], au même titre que le λ-calcul[Alo41][AC98] est considéré comme le fondement de la programmation fonctionnelle. En particulier, le π-calcul permet de rendre compte

Opérateur	**Notation**	**Description informelle**
Composition séquentielle	**exit** $(v_1, ...v_m) >>$ **accept** $x_1 : T_1, ...x_m : T_m$ **in** P	Le processus qui a terminé avec succès transmit des valeurs $v_1...v_m$ au processus suivant qui peut les consulter à travers les variables $x_1 : T_1, ...x_m : T_m$ respectivement.
Choix généralisé	**choice** $x : Type$ [] $P(x)$	Si P dépend des variables x, l'opérateur offre le choix entre les processus pour toutes les valeurs possibles de $x \in Type$.
Déclaration de variable	**let** $x : Type = v$ **in** P	Réalise l'instanciation de la variable x avec la valeur v dans P.
Prédicat de garde	$[pred] -> P$	Le processus aura le comportement de P si *pred* est vrai, sinon il devient **stop**.

TAB. 2.4 – *Principaux opérateurs LOTOS avec utilisation de données*

$exit(v_1...v_m) \ \underrightarrow{\delta!(valeur(v_1)...valeur(v_n))} \ stop$
$g!v; P \ \underrightarrow{g : (valeur(v))} \ P \quad (g \in G)$ $g?y : T; P \ \underrightarrow{g?(p)} \ \textbf{let} \ \ x : T = p \ \textbf{in} \ \ P(g \in G)(p \in T)$
$\dfrac{P \xrightarrow{g} P'}{\textbf{let } x : T = v \textbf{ in } P \xrightarrow{g} \textbf{let } x : T = v \textbf{ in } P'} \qquad \dfrac{[v/x]P \xrightarrow{g} P'}{\textbf{let } x : T = v \textbf{ in } P \xrightarrow{g} P'}$
$\dfrac{P \xrightarrow{g} P' \quad valeur(v) = true}{[v] -> P \xrightarrow{g} P'}$
$\dfrac{P_X[g_1/g'_1...g_n/g'_n](valeur(v_1)/y_1...valeur(v_n)/y_n) \xrightarrow{g} Q' \quad X[g'_1...g'_n](y_1 : T_1...y_n : T_n) = P_X}{X[g_1...g_n](valeur(v_1)...valeur(v_n)) \xrightarrow{g} Q'}$

TAB. 2.5 – *Sémantique opérationnelle de LOTOS*

la notion de migration au cours de calcul, il permet de plus de décrire des systèmes de processus mobiles, c'est à dire. des systèmes dont le nombre de processus ainsi que liens de communication entre processus peuvent varier au cours du temps : les noms de canaux sont transmis comme valeurs sur des canaux existants, ce qui permet à un processus de recevoir un canal nouvellement créer et d'acquérir ainsi de nouvelles possibilités de communication. Le π-calcul est principalement utilisé pour établir des preuves d'équivalence entre des modèles de systèmes distribués[Mil99]. Nous présentons ce calcul de systèmes communicants dans lequel non seulement les processus ont la structure dynamique, mais aussi les agents composants le système, peuvent porter l'information qui change ces liens.

Dans cette section on présente le π-calcul dans les deux formes monadique et polyadique : syntaxe, exemples de base, sémantique par réduction et par système de transitions étiquetées, relations de bisimulations et théorie algébrique.

Commençons par une série d'exemples, qui ont une signification pratique. Dans le premier exemple, nous le présentons d'abord dans CCS et nous proposons un diagramme (graphique de flux), qui représente les liens entre les agents.

Supposons qu'un agent P veut envoyer la valeur 5 à un agent R, via le nom a, et que R désire recevoir n'importe quelle valeur sur ce nom. Alors le graphique de flux approprié est dans la figure 2.7

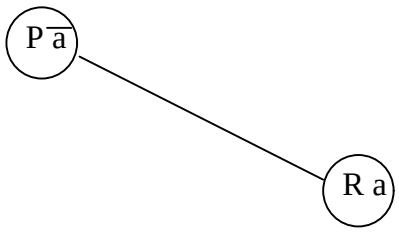

FIG. 2.7 – Canal de communication entre deux processus

Nous pouvons avoir, par exemple, $P \equiv \overline{a}5.P'$ et $R \equiv a(x).R'$. Le préfixe $a(x)$ lie la variable x dans R', en général nous employons des parenthèses pour indiquer la présence d'une variable. Le système dépeint dans le graphique de la figure 2.7 est représenté par l'expression

$$(\overline{a}5.P' \mid a(x).R')/a$$

L'opérateur $/a$ est appelé une restriction, la liaison a est privée à P et R.

On suppose maintenant que P veut déléguer à un nouveau agent, Q, la tâche de transmettre 5 à R. Nous supposons donc que P est connecté à Q au départ par une liaison b.

Soit le processus $P \equiv \overline{b}a.\overline{b}5.P'$, il envoie via la liaison b, la liaison a et la valeur 5. Et soit $Q \equiv b(y).b(z).\overline{y}z.0$, il reçoit une liaison et une valeur sur b, ensuite il transmet la valeur via

la nouvelle liaison reçue et se termine. Notez que le nom a n'est pas dans l'expression Q. Q ne possède aucune liaison à R au départ. Le système entier est maintenant :

$$(\overline{b}a.\overline{b}5.P' \mid b(y).b(z).\overline{y}z.0 \mid a(x).R') \diagup a \diagup b$$

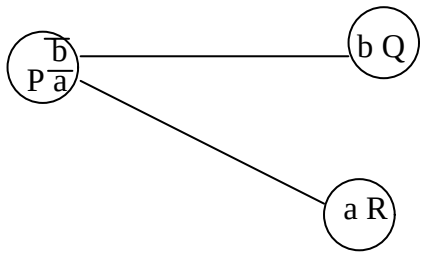

FIG. 2.8 – Mobilité des canaux de communication 1

Après deux communications, sur la liaison b, le système devient :

$$(P' | \overline{a}5.0 | a(x).R') \diagup a \diagup b$$

Si a n'apparaît pas dans P', la nouvelle configuration du système est donnée par la figure 2.9, indiquant que la liaison a de P s'est déplacée à Q, et Q est devenu $Q' \equiv \overline{a}5.0$.

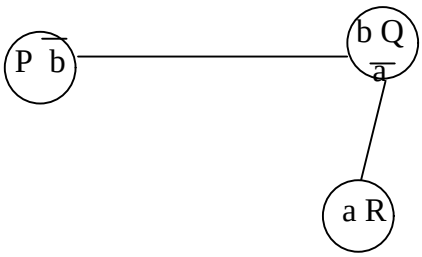

FIG. 2.9 – Mobilité des canaux de communication 2

Ce formalisme, dans lequel les noms des liaisons apparaissent comme des paramètres dans la communication, va au-delà de CCS. Avec les variables sur des noms de liaison, aussi bien que sur des valeurs de données ordinaires, le calcul deviendrait riche de primitives. Mais toute distinction entre des noms de liaison, des variables et des valeurs de données ordinaires est enlevée. Il y aura juste deux classes essentielles d'entité : noms et des agents.

2.2.1 π-calcul monadique

Syntaxe

Dans cette section on introduit la syntaxe du π-calcul. L'entité primitive dans le π-calcul est le nom (canal).

· Soit $\mathcal{X}$ l'ensemble des noms parcouru par $x, y, ...$

· Soit $\mathcal{P}$ l'ensemble des identifiants de processus parcouru par $P, Q, ...$

La sysntaxe formelle de π-calcul est donnée par la figure 2.10 :

$P, Q, R ::=$	**Processus**
$\Sigma\pi_i.P_i$	
$\mid P\mid Q$	*Définition local*
$\mid\ !P$	*Composition parallèle*
$\mid (vn)P$	*Réstriction*

FIG. 2.10 – *Syntaxe de Pi-calcul*

1. La sommation $\sum_{i\in I}\pi_i.P_i$, I ensemble d'index fini.

- Pour l'ensemble vide I=ϕ, la somme devient 0.

- Dans le cas $\pi.P$, le préfixe π représente une action atomique. Ce préfixe (action) peut avoir les formes suivantes :

$x(y)$ Action d'entrée signifie que P peut recevoir n'importe quel nom w, sur le canal x et se transformer ensuite en $P\{w/y\}$.

$\overline{x}.y$ Action de production c'est à dire on peut émettre le nom libre y sur le canal x.

τ Représente l'action silencieuse.

Dans les deux cas on appel x sujet et y l'objet de l'action.

2. La composition $P|Q$: Les deux processus sont activés simultanément, donc indépendamment, mais ils peuvent communiquer.

3. L'opérateur de duplication $!P$: donne plusieurs copies parallèles de P, $P|P|...$

4. La restriction $(vx)P$: seul le processus P a accès sur le nom x.

Définitions de base

- Les noms libres fn(P) de P sont les noms qui figurent dans P, et ils ne sont pas liés par un préfixe d'entrée ou par une restriction. Et en les appellent noms liés bn(P) dans les autres cas.

$$\begin{aligned} \text{bn}(x(y)) &= \{y\}, \qquad \text{fn}(x(y)) = \{x\}. \\ \text{bn}(\overline{x}y) &= \phi, \qquad\ \ \text{fn}(\overline{x}y) = \{x, y\}. \end{aligned}$$

Exemples

- Soit le processus : $\overline{x}y.0 \mid x(u).\overline{u}v.0 \mid \overline{x}z.0$

  Se transforme en : $\overline{x}y \mid x(u).\overline{u}v \mid \overline{x}z$ ce processus est de la forme $P|Q|R$. Une des deux communications (mais pas les deux) peut se faire à travers le canal x, P envoi y à Q, ou R envoi z à Q. Donc on obtient :

$$0|\overline{y}v|\overline{x}z \quad \text{ou} \quad \overline{x}y|\overline{z}v|0$$

- Soit le processus $(vx)(\overline{x}y \mid x(u).\overline{u}v) \mid \overline{x}z$

  Dans ce cas le nom x qui est libre dans R est différent de x qui est lié dans P et Q, donc on a une seule communication :

$$0|\overline{y}v|\overline{x}z$$

2.2.2 π-calcul polyadique

π-calcul polyadique[Mil93], Afin de faciliter la manipulation des termes du π-calcul, on va utiliser une forme moins primitive pour la communication sur un canal. On va permettre le passage simultané de plusieurs arguments sur un canal. La syntaxe est donc la même que celle de la forme monadique cependant les changements suivants sont à considérer :

· Le processus $x(\widetilde{y}).P$, attend un tuple $\widetilde{z}$ soit transmis sur le canal x, puis il continu le processus P avec la substitution de tuple $\widetilde{y}$ par le tuple $\widetilde{z}$.

· Le processus $\overline{x}\widetilde{y}.P$, envoie le tuple $\widetilde{y}$ sur le canal x, et ensuite continu le processus P.

2.2.3 Sémantique opérationnelle du π-calcul

Traditionnellement, la sémantique d'une algèbre de processus est donnée, en termes de système de transitions étiquetées, décrivant les évolutions possibles des processus. Milner offre dans [MPW92] une directive pour la définition des systèmes de réduction dans des algèbres de processus, où les axiomes pour la relation de congruence structurelle sont présentés avant le système de réduction.

La sémantique opérationnelle de π-calcul, est représentée par des systèmes de transitions étiquetées(STE) [MPW92], ou un système réduit avec une relation de congruence structurelle ($\equiv$)[MPW92], c'est la plus petite relation de congruence sur les processus.

Sémantique de réduction

Définition 2.2 *(congruence structurelle) : Dans un système de réduction, on définit une congruence structurelle notée $\equiv$ telle que les opérateures $+$ et $\mid$ sont associatif et commutatif.*

Cette congruence c'est la plus petite relation de congruence sur $\mathcal{P}$, qui vérifier les règles suivantes :

1. Les agents (processus) sont identifiés, s'ils diffèrent seulement par un changement de noms liés.

2. $(\mathcal{X}/\equiv, +, 0)$, est un monoïde symétrique.

3. $(\mathcal{P}/\equiv, +, 0)$, est un monoïde symétrique.

4. $!P \equiv P|!P$

5. $(vx)0 \equiv 0, (vx)(vy)P \equiv (vy)(vx)P$

6. si $x \in fn(P)$ *alors* $(vx)(P|Q) \equiv (vx)P|(vx)Q$

Règles de réduction Une relation de réduction $P \longrightarrow P'$ sur un processus P. Signifie que P peut ce transformer en P' par une étape de calcul simple. Donc toute étape de calcul, c'est une interaction entre deux termes. La première réduction c'est la communication :

$$\textbf{COMM} \ :(... + x(y).P \mid ... + \overline{x}z.Q \ \rightarrow \ P\{z/y\} \mid Q \ )$$

Elle représente une communication entre deux processus atomiques $\pi.P$ qui sont complémentaires. Les (...) peuvent être **0**, ou d'autres communications.

COMM c'est un axiome pour la réduction, les autres réductions sont des règles d'inférence.

$$\textbf{PAR} : \quad \frac{P \rightarrow P'}{P|Q \rightarrow P'|Q} \qquad \textbf{RES} : \frac{P \rightarrow P'}{(vx)P \rightarrow (vx)P'}$$

$$\textbf{STRUCT} : \frac{Q \equiv P \quad P \rightarrow P' \quad P' \equiv Q'}{Q \rightarrow Q'}$$

Remarque : on ne peut pas réduire dans les cas suivants

1. Si on a un préfixe, par exemple :

$$u(v).(x(y) \mid \overline{x}z)$$

L'opération de préfixage est prioritaire par rapport à l'opération de communication.

2. Si on a une réplication.

Si on a $P \longrightarrow P'$ alors, au lieu de la réduction, $!P \longrightarrow !P'$, qui est équivalente à la réduction illimitée de copies de P, nous pouvons toujours réduire de la manière suivante :

$$!P \equiv P|P|...|P|!P \longrightarrow^n P'|P'|...|P'|!P'$$

Ainsi (après n réductions) réduisant autant de copies de P que nous exigeons.

Sémantique de transitions étiquetées

Action : la transition dans π-calcul est de la forme :

$$P \xrightarrow{a} Q$$

Intuitivement, cette transition représente une transformation de processus P vers le processus Q en consommant l'action a. Dans le π-calcul il y a quatre forme de l'action a :

1. L'action silencieuse τ, comme dans CCS, signifie que P se transforme en Q , avec aucune interaction avec l'environnement. Les actions silencieuses peuvent être naturellement le résultat des agents de la forme $\tau.P$, mais aussi des communications dans un agent.

2. Une action de production libre $\overline{x}y$, la transition $P \xrightarrow{\overline{x}y} Q$ implique que P peut émettre le nom libre y sur le canal x . Les actions de production libres résultent de la forme de préfixe de production $\overline{x}y.P$.

3. Une action d'entrée $x(y)$. Intuitivement, $P \xrightarrow{x(y)} Q$ signifie que P peut recevoir n'importe quel nom w, sur le canal x et se transformer ensuite en $Q\{w/y\}$. Ici y représente une référence à la place où le nom reçu sera substitué. Les actions d'entrée résultent de la forme de préfixe d'entrée $x(y).P$.

4. Une action de production liée $\overline{x}(y)$. Intuitivement, $P \xrightarrow{\overline{x}(y).} Q$ signifie que P émet un nom privé sur le canal x , et (y) est une référence où ce nom privé arrive. Comme dans l'action d'entrée ci-dessus, y est incluse entre parenthèses pour souligner que c'est une référence et ne représente pas de nom libre.

La relation de transition c'est la plus petite relation qui satisfait les règles du tableau 2.6, page 38 :

2.2.4 Exemple

Passage de liaison[MPW92] : Soit le graphique dans la figure 2.11 :

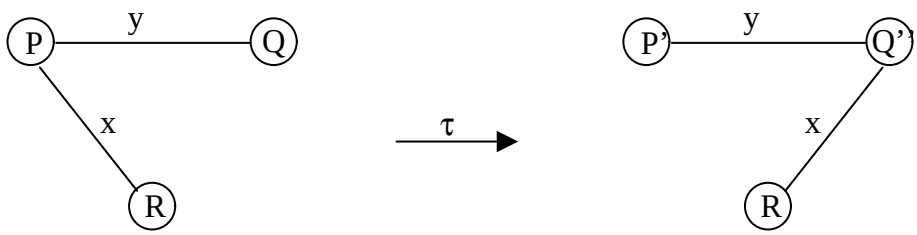

FIG. 2.11 – Passage de liaison 1

L'agent P est lié à l'agent R par le canal x, et désir envoyer ce canal à Q à travers le canal y. Q prêt de le recevoir. Ainsi P peut être $\overline{y}x.P'$ et Q peut être $y(z).Q'$, dans ce cas la transition est :

TAU-ACT : $\dfrac{-}{\tau.P \xrightarrow{\tau} P}$ OUTPUT-ACT : $\dfrac{-}{\overline{x}y.P \xrightarrow{\overline{x}y} P}$
INPUT-ACT : $\dfrac{-}{x(z).P \xrightarrow{x(w)} P\{w/z\}}$ $w \notin \text{fn}((z)P)$
SUM : $\dfrac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'}$ MATCH : $\dfrac{P \xrightarrow{\alpha} P'}{[x = x]P \xrightarrow{\alpha} P'}$
IDE : $\dfrac{P\{\widetilde{y}/\widetilde{x}\} \xrightarrow{\alpha} P'}{A(\widetilde{y}) \xrightarrow{\alpha} P'}$ $A(\widetilde{x}) \stackrel{def}{=} P$
PAR : $\dfrac{P \xrightarrow{\alpha} P'}{P\|Q \xrightarrow{\alpha} P'\|Q}$ $\text{bn}(\alpha) \cap \text{fn}(Q) = \phi$
COM : $\dfrac{P \xrightarrow{\overline{x}y} P' \quad Q \xrightarrow{x(z)} Q'}{P\|Q \xrightarrow{\tau} P'\|Q'\{y/z\}}$ CLOSE : $\dfrac{P \xrightarrow{\overline{x}(w)} P' \quad Q \xrightarrow{x(w)} Q'}{P\|Q \xrightarrow{\tau} (vw)(P'\|Q')}$
RES : $\dfrac{P \xrightarrow{\alpha} P'}{(vy)P \xrightarrow{\alpha} (vy)P'}$ $y \notin \text{n}(\alpha)$ OPEN : $\dfrac{P \xrightarrow{\overline{x}y} P'}{(vy)P \xrightarrow{\overline{x}(y)} P'\{w/y\}}$ $y \neq x$ $w \notin \text{fn}((y)P')$

TAB. 2.6 – Sémantique opérationnelle de pi-calcul

$$\overline{y}x.P'|y(z).Q'|R \xrightarrow{\tau} P'|Q'\{x/z\}|R$$

Donc Q" dans la figure 2.11 est $Q'\{x/z\}$. Cette figure illustre le cas dans lequel $x \notin \text{fn}(Q)$, signifiant que Q ne possède aucune liaison x avant la transition. Mais la transition c'est la même si $x \in \text{fn}(Q)$, il n'y a aucune raison que Q ne doit pas recevoir une liaison qu'il possède déjà. La figure illustre aussi le cas dans lequel $x \notin \text{fn}(P')$, signifiant que P' n'a aucune x-liaison après la transition, mais de nouveau cette condition n'affecte pas la transition.

La situation n'est pas différente quand la liaison y entre P et Q est privée. Dans ce cas le graphe de flux approprié est dans la figure 2.12 :

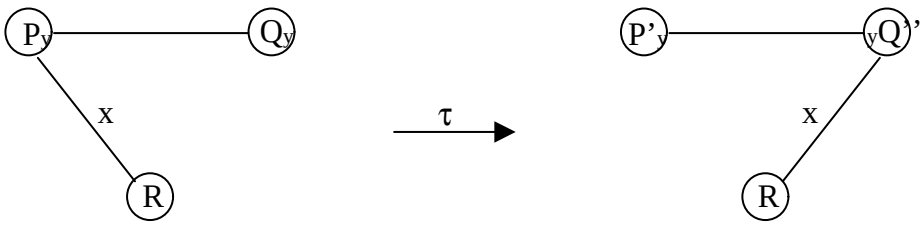

FIG. 2.12 – Passage de liaison 2

La liaison y (privée) est représentée par une restriction, donc la transition est maintenant

$$(vy)(\overline{y}x.P'|y(z).Q')|R \xrightarrow{\tau} (vy)(P'|Q'\{x/z\})|R$$

En conclusion, en $\pi - calcul$, les processus communiquent de manière synchrone, point à point, à travers des canaux (qui peuvent être des canaux privés). La topologie de communication est dynamique. Le calcul est muni d'une sémantique mathématique bien fondée, tout en restant à la fois simple et expressif. Il permet de coder le $\lambda - calcul$ [Mil90]et les structures de données, et il a été utilisé pour prouver des propriétés de correction des protocoles (par exemple le protocole PLMN pour le GSM[OP92]).

2.3 Variantes du π-calcul

De nombreuses variantes de π-calcul ont été proposées et étudiées :

2.3.1 π-calcul distribué (Dπ)

Présentation informelle du Dπ-calcul

Dπ-calcul (pour distributed π-calcul)[HR02][Hen07] est un calcul de processus introduit par Matthew Hennessy et James Riely pour étudier la répartition du calcul. Un calcul est modélisé en Dπ par des processus. Afin de représenter la répartition du calcul, Dπ repose sur la notion

de localités, ou de sites, de sorte que tous les processus sont annotés par la localité dans laquelle ils s'exécutent :

$$l[\![P]\!]$$

désigne le processus P placé dans la localité l.

Ces processus localisés peuvent être composés en parallèle. A titre d'exemple :

$$l[\![P]\!] \mid k[\![Q]\!]$$

Comporte un processus P s'exécutant dans la localité l et un processus Q s'exécutant indépendamment dans la localité k.

À chaque instant, un des processus d'un système peut réaliser une étape de calcul interne ou bien interagir : une interaction met en jeu deux processus qui décident de se synchroniser et d'échanger un message. Pour avoir lieu, cette communication suppose qu'il existe un canal sur lequel un des deux processus émet un message tandis que l'autre processus écoute. Puisque la synchronisation entre deux processus est une opération complexe, Dπ impose que les communications soient locales, c'est-à-dire que les canaux sur lesquels elles s'effectuent soient eux aussi localisés et ne soient accessibles qu'aux processus s'exécutant dans leur localité. Soit le processus

$$c! < m > stop$$

qui procède à l'émission du message m sur le canal c puis s'arrête, et le processus « inverse »

$$c? < x > stop$$

qui procède à la réception d'un message sur le même canal, peuvent être composés de la façon suivante :

$$l[\![c! < m > stop]\!] \mid l[\![c? < x > stop]\!]$$

Ces deux processus étant placés dans la même localité, ils peuvent se synchroniser et s'échanger le message m sur le canal c. Le système qu'ils forment peut alors se réduire à :

$$l[\![c! < m > stop]\!] | l[\![c? < x > stop]\!] \xrightarrow{\tau} l[\![stop]\!] | l[\![stop]\!]$$

Cet exemple illustre uniquement le mécanisme de synchronisation, puisque les deux processus s'arrêtent dès que le message est échangé. Mais on peut bien entendu aussi décrire des processus capables de poursuivre leur calcul après une communication. Ainsi, $c! < m >$ peut préfixer n'importe quel processus P. Le processus $c! < m >$ commence son calcul en émettant le message m avant de déclencher sa continuation P, quels que soient les calculs que P effectue. Symétriquement $c?(x)Q$ attend un message sur le canal c pour poursuivre son exécution

définie par Q. Mais, pour qu'il y ait bien communication et pas seulement synchronisation, le processus récepteur Q doit prendre le contenu du message. Le préfixe $c?(x)$ lie la variable x qui désignera dans le processus Q le message reçu au cours de cette communication, exactement comme les paramètres d'une fonction dans un langage de programmation. Lorsque la fonction est appelée, ses arguments formels sont remplacés par les arguments réels à la CCS.

L'indépendance des processus entre eux implique que cette communication pourra avoir lieu indépendamment les autres processus présents, par exemple :

$$l[\![c! < m > P]\!]|k[\![R_1]\!]|l[\![c? < x > Q]\!]|l[\![R_2]\!] \xrightarrow{\tau} l[\![P]\!]|k[\![R_1]\!]|l[\![Q\{m/x\}]\!]|l[\![R_2]\!]$$

Cette situation plus complexe illustre également que la proximité syntaxique des deux processus qui interagissent dans l'exemple précédent n'est pas indispensable.

Si ces deux primitives de communication étaient les seules opérations permises pour les processus, Dπ ne permettrait de décrire comme systèmes répartis que des systèmes dans lesquels les processus sont distribués une fois pour toute sur l'ensemble des localités existantes. Dπ fournit par conséquent d'autres primitives pour décrire le calcul. Dπ introduit de la dynamique notamment en permettant aux processus de migrer d'une localité à l'autre. Puisque la localisation prend la forme très simple d'une annotation indiquant le nom de la localité dans laquelle le processus s'exécute, l'opération de migration s'exprime aisément par :

$$l[\![goto\ k.\ P]\!] \longrightarrow k[\![P]\!]$$

Un autre aspect dynamique de Dπ est la possibilité de générer de nouveaux noms, qu'il s'agisse de noms de localités ou de canaux.

2.3.2 π_{1l}-calcul

Le π_{1l}-calcul[AP94][Ama97][Ama00], extension du π-calcul asynchrone avec primitives de localité et un modèle de défaillance. Dans π_{1l}, les domaines sont des localités abritant des processus. L'organisation des localités est plate (un seul niveau de hiérarchisation des localités). Chaque localité dispose d'un unique processus de contrôle contenant l'état actif ou en panne de la localité et constitue un domaine de défaillances : une localité en panne ne peut plus émettre de messages ou accepter de processus en cours de migration. Le comportement de contrôle du domaine se limite à observer son état (ping), ou le faire tomber en panne de manière irréversible (stop). La portée des récepteurs est répartie, ce qui les rend accessibles depuis n'importe quelle localité. La communication distante est primitive et le routage des messages est implicite. Outre la migration des messages, les primitives de mobilité se limitent à déclencher l'exécution d'un processus dans une localité distante active (spawn).

2.3.3 $b\pi$-calcul

Le $b\pi$-calcul[Ene01] est un calcul de processus mobiles qui communiquent seulement par diffusion. Il est inspiré à la fois du CBS (Calculus of Broadcasting Systems)[Pra95] en considérant la diffusion comme seule primitive de communication, et du π-calcul, du quel il hérite

la même syntaxe et surtout la même manière d'exprimer la mobilité. Il diffère de CBS par le fait que les communications utilisent des canaux (ou ports) et les valeurs transmises sont aussi des canaux.

2.3.4 $\pi - calcul$ d'ordre supérieur

Le $\pi - calcul$ d'ordre supérieur[San92], ou HOπ, étend le π-calcul en permettant non seulement des noms mais aussi des processus a être transmis comme valeurs, rendant ainsi la mobilité des processus plus explicite. Même si le HOπ, semble beaucoup plus expressif que l'ordinaire (premier ordre)π-calcul, un point important est qu'il peut être fidèlement encodé dans le π-calcul[San92].

2.3.5 $\pi - calcul$ asynchrone

Le $\pi - calcul$ asynchrone [HT91][Bou92][ACS96], ou π_a,est une variante du π-calcul, où le préfixe de sortie est toujours de la forme $\overline{u}\widetilde{x}.0$. La communication est asynchrone dans le sens où il n'est pas possible pour l'émetteur de déterminer quand la production est consommée par une entrée. A titre d'exemple le π-calcul asynchrone a été utilisé avec succès pour confirmer formellement la validité des techniques d'optimisation dans le langage de programmation Pict[Pie98], il a été démontré qu'il est strictement moins expressif que le standard (synchrone) π-calcul[Pal03][HT92].

2.3.6 Fusion calculus

Fusion calcul[Vic98][PV98], à été défini dont l'objectif de donner un calcul canonique de concurrence. C'est une extension simplifiée du π-calcul. Nous mentionnons trois indications de sa plus grande expressivité :

- Dans π-calcul la modélisation des états partagés, des formalismes des contraintes concurrentes, est représentée par l'introduction d'un processus supplémentaire. Par contre dans fusion calcul les mécanismes de base des mises à jour d'un état partagé sont déjà présents, et ça modélisation est simple.

- Les réductions fortes du λ-calcul, sont difficiles à encoder dans le π-calcul. Ces réductions peuvent être représentées facilement dans fusion calcul.

- Le polyadique π-calcul est un sous calcul de fusion calcul. Donc tout ce qui peut être fait dans le polyadique π-calcul peut être fait dans le fusion calcul sans complications ajoutées.

2.3.7 $\pi - calcul$ réceptif

La notion de réceptivité[San97][SW01] se traduit par : le canal x est réceptif dans le processus P si à tout moment, P est prêt à recevoir sur x. Ainsi, tout message sur un canal réceptif doit

pouvoir être directement consommé par un processus. Permet les avantages de la réceptivité des canaux : ces canaux peuvent être implémentés efficacement relativement aux canaux standards. L'autre avantage concerne les propriétés algébriques, parmi lesquelles la copie, la distributivité, la τ-insensibilité et les lois de la parallélisation.

2.3.8 $\pi - calcul$ réceptif distribué

Le $\pi - calcul$ réceptif distribué[ABL99], est vu comme une simplification de π-calcul distribué, et extension du π-calcul asynchrone et polyadique, comportant des notions explicites de localité et de migration, dans ce calcul la communication est purement locale. La disponibilité des canaux est assurée par la propriété de réceptivité[San97].

2.4 Join-Calcul

Le join-calcul[FGJ+96][Fou98], s'inspire largement de deux modèles bien connus du parallélisme, les calculs des processus CCS ou le $\pi-$calcul d'une part, et la machine chimique abstraite[BB92] de Berry et Boudol d'autre part. Il correspond au noyau d'un langage de programmation directement utilisable, en particulier, il est implémenté de manière répartie. Il préserve la plupart des propriétés formelles du π-calcul[Fou98]. Le join-calcul est un modèle de spécification formelle des applications asynchrones, distribuées, avec la notion de mobilité.

2.4.1 Machine chimique abstraite

Dans[BB92], les auteurs proposent un modèle nommé : machine abstraite chimique (CHAM), la sémantique chimique utilise un ensemble de règles chimiques, sur des multi-ensembles de molécules (termes). Les règles chimiques fonctionnent sur les multi-ensembles des termes (solutions chimiques) :

- – Les règles structurelles ($\rightleftharpoons$) sont réversibles ; ils représentent des réarrangements syntaxiques de termes dans la solution.
  – Les règles de réduction ($\rightarrow$) consomment quelques termes spécifiques dans la solution chimique, les remplaçant par d'autres termes, ils correspondent aux pas de calcul de base.

Pour illustrer l'approche chimique, nous prenons l'exemple de passage d'une valeur suivant :

$$\textbf{Str-join} \qquad P|P' \rightleftharpoons P, P'$$

$$\textbf{Red} \qquad \overline{x} < \widetilde{v} > |x < \widetilde{y} > .P \rightarrow P\{\widetilde{v}/\widetilde{y}\}$$

$P, Q, R ::=$		**Processus**
	$x < v_1, ..., v_n >$	*Message asynchrone*
	$\mid$ **def** D **in** P	*Définition local*
	$\mid\ P \mid Q$	*Composition parallèle*
	$\mid$ **0**	*Processus inerte*
$D ::=$		**Définition**
	$J \rhd P$	*Règle réactive*
	$\mid\ D \wedge D\prime$	*Composition*
	$\mid\ \top$	*Définition vide*
$J ::=$		**Join-patterns**
	$x < y_1, ..., y_n >$	*Patterns message*
	$\mid\ J \mid J\prime$	*Pattern de join*

FIG. 2.13 – *Syntaxe de Join-calcul*

Str-join : chaque molécule de la forme $P|P'$ peut se partitionner en deux petite molécules P et P'. Et inversement chaque paire de molécules P et P' peut devenir une seule molécule $P|P'$.

Red : c'est une règle de réduction qui consomme la seule molécule qui contient deux processus en composition parallèle, et elle produit une seule molécule.

Par exemple on a les étapes chimiques suivantes

$$\begin{aligned}\{\overline{x} < 1 > |P|x < u > .Q\} &\rightharpoonup\!\rightharpoonup \{\overline{x} < 1 >, P, x < u > .Q\} \\ &\leftharpoondown \{P, \overline{x} < 1 > |x < u > .Q\} \\ &\rightarrow \{P, Q\{1/u\}\} \\ &\rightharpoonup \{P|Q\{1/u\}\}\end{aligned}$$

2.4.2 Syntaxe

Soit $\mathcal{N}$ un ensemble infini de noms des ports (les ports sont appelés aussi des canaux) parcouru par x, y,...

Il y a trois sortes d'expressions, des processus, des définitions et des join-patterns (filtres), (voir figure 2.13).

Un processus P peut être soit l'envoi asynchrone d'un message $x < v_1, ..., v_n >$, soit la définition local de nouveaux noms **def** D **in** P, soit une composition parallèle de processus $P|Q$, ou le processus inerte **0**. Une définition D peut être soit la définition vide $\top$, soit une composition de définition D, D' connectées par l'opérateur $\wedge$, soit une règle de réaction $J \rhd P$, consommant les messages correspondant au filtre J pour exécuter P. Les filtres J modélisent la réception et la synchronisation de messages : un filtre de message $x < \widetilde{y} >$ attend qu'un

$$
\begin{array}{rcl}
\mathrm{fv}[x < v_1, ..., v_n >] & \stackrel{def}{=} & \{x, v_1, ..., v_n\} \\
\mathrm{fv}[\textbf{def } D \textbf{ in } P] & \stackrel{def}{=} & (\mathrm{fv}[P] \cup \mathrm{fv}[D]) \backslash \mathrm{dv}[D] \\
\mathrm{fv}[P|P'] & \stackrel{def}{=} & \mathrm{fv}[P] \cup \mathrm{fv}[P'] \\
\mathrm{fv}[\mathbf{0}] & \stackrel{def}{=} & \phi \\
 & & \\
\mathrm{fv}[J \rhd P] & \stackrel{def}{=} & \mathrm{dv}[J] \cup (\mathrm{fv}[P] \backslash \mathrm{rv}[J]) \\
\mathrm{fv}[D \wedge D\prime] & \stackrel{def}{=} & \mathrm{fv}[D] \cup \mathrm{fv}[D\prime] \\
\mathrm{fv}[T] & \stackrel{def}{=} & \phi \\
 & & \\
\mathrm{dv}[J \rhd P] & \stackrel{def}{=} & \mathrm{dv}[J] \\
\mathrm{dv}[D \wedge D'] & \stackrel{def}{=} & \mathrm{dv}[D] \cup \mathrm{dv}[D'] \\
\mathrm{dv}[T] & \stackrel{def}{=} & \phi
\end{array}
$$

$$
\begin{array}{rcl@{\qquad}rcl}
\mathrm{dv}[x < y_1, ..., y_n >] & \stackrel{def}{=} & \{x\} & \mathrm{dv}[J|J'] & \stackrel{def}{=} & \mathrm{dv}[J] \uplus \mathrm{dv}[J'] \\
\mathrm{rv}[x < y_1, ..., y_n >] & \stackrel{def}{=} & \{y_1, ..., y_n\} & \mathrm{rv}[J|J'] & \stackrel{def}{=} & \mathrm{rv}[J] \uplus \mathrm{rv}[J']
\end{array}
$$

FIG. 2.14 – *Les portées dans Join-calcul*

message $\widetilde{y}$ sur le canal x soit présent, le filtre $J|J'$ attend que les filtres J et J' soient satisfaits pour être satisfait.

Les noms de ports définis sont récursivement liés dans toute la définition def D in P, c'est-à-dire dans le processus principal P et dans les processus gardés sous la définition D. On note les variables reçues rv[J], variables définies dv [J] et dv [D], et variables libre fv[D] et fv[P] sont formellement définies pour le calcul dans la figure 2.14.

Sémantique opérationnelle

Solution réflexive : la solution chimique comporte deux parties de molécules de multi-ensemble ($\mathcal{D} \vdash \mathcal{P}$). La molécule $\mathcal{P}$ représente des processus en cours d'exécution parallèle, $\mathcal{D}$ c'est l'ensemble des règles réactives. La définition active D représente des règles réactives définies des réductions possibles pour les processus, les processus P, représentent l'état du calcul, il peut introduire de nouveaux noms et règles de réactions, ce qui signifie la machine réflexive, les règles chimiques pour la RCHAM sont définies dans la figure 2.15, page 46.

La sémantique donnée dans la figure 2.15, c'est une collection de règles chimique applicables sur un fragment de solution réflexive.

Les quatre premières règles structurelles avec les deux opérateurs | et $\wedge$, sont associative et commutative, avec des unités 0 et $\top$. La réduction simple **Red** illustre l'utilisation des règles active $J \rhd P$ apparue dans la partie gauche de la solution chimique. La règle réaction $J \rhd P$ est inchangeable à gauche, et peut être employée plus tard pour effectuer des nouveaux

$$
\begin{array}{lrcl}
\textbf{str-join} & \vdash P_1|P_2 & \rightleftharpoons & \vdash P_1, P_2 \\
\textbf{str-null} & \vdash 0 & \rightleftharpoons & \vdash \\
\textbf{str-and} & D_1 \wedge D_2 \vdash & \rightleftharpoons & D_1, D_2 \vdash \\
\textbf{str-nodef} & T \vdash & \rightleftharpoons & \vdash \\
\textbf{str-def} & \vdash \textbf{def } D \textbf{ in } P & \rightleftharpoons & D\sigma_{dv} \vdash P\sigma_{dv}
\end{array}
$$

$$
\begin{array}{lrcl}
\textbf{Red} & J \rhd P \vdash J\sigma_{rv} & \rightarrow & J \rhd P \vdash P\sigma_{rv}
\end{array}
$$

Condition pour la substitution

str-def σ_{dv} instanciation des variables de port dv[D] a des noms distincts, frais : $dom(\sigma_{dv}) \cap \text{fv}[\mathcal{S}] = \phi$ où $\mathcal{S}$ est la solution initiale,

red σ_{rv} Substitue les noms transmis aux variables distinctes reçues rv [J]

FIG. 2.15 – *La machine chimique réflexive*

pas de réduction.

2.4.3 Localité, migration et pannes

Calcul avec location

Dans join-calcul la répartition des ressources est organisée par emplacements. Intuitivement, un emplacement réside sur un site particulier, et peut contenir des processus et des règles de réductions. Un emplacement peut se déplacer d'un site à l'autre, ainsi un emplacement peut représenter un agent mobile. Un emplacement peut également contenir des sous-emplacements, ce qui donne une structure hiérarchique au calcul et permet de modéliser les sites comme des emplacements particuliers.

Les noms d'emplacements sont des valeurs de première classe, tout comme les noms de canaux. Ces noms peuvent être communiqués à d'autres processus éventuellement dans d'autres places, ce qui permet de programmer la gestion des places tout en contrôlant la migration par la portée lexical.

Solution distribuée (Plusieurs machines chimiques)

Afin de modéliser la présence de plusieurs sites de calcul, la machine abstraite chimique réflexive a été raffiner, chaque emplacement contient une machine abstraite, et l'ajout d'une règle de communication asynchrone entre emplacements : Ainsi, l'état du calcul est maintenant représenté par une famille de paires de multi-ensembles $\{\mathcal{D}_i, \mathcal{P}_i\}$ qui contiennent respectivement les règles de réactions et les processus en cours d'exécution dans chaque emplacement.

Informellement, ce nouveau mécanisme de calcul reflète le *routage* des messages d'un point à l'autre du réseau.

$$\textbf{COMM} \quad \vdash x < \widetilde{v} > \,||\, J \rhd P \vdash \quad \rightarrow \quad \vdash \,||\, \quad J \rhd P \vdash x < \widetilde{v} > \quad (x \in \mathrm{dv}[J])$$

L'arbre de location Soit un ensembles de noms de location noté par $\mathcal{X}$, les lettres $a, b, ... \in \mathcal{X}$, sont utilisées pour les noms de location, et $\varphi, \psi ... \in \mathcal{X}^*$, sont utilisées pour des chaînes finis, pour les noms de location. Comme les noms des ports, les noms de locations peuvent être créer localement, envoyer et recevoir des message.

La nouvelle syntaxe avec location, est définie en rajoutant le constructeur :

$$D \stackrel{def}{=} ...|a[D' : P]$$

Informellement, la définition $a[D\prime : P]$correspond à la solution local $\{D'\} \vdash_{\varphi a} \{P\}$.

La sémantique de ce nouveau constructeur est, la création d'une sous-location de la location courante, qui contient initialement l'unique définition D et l'unique processus P en cours d'exécution. Plus précisément, on a la nouvelle règle structurelle :

$$\textbf{Str-loc} \quad a[D : P] \vdash_{\varphi} \quad \rightleftharpoons \quad \vdash_{\varphi} ||\{D\} \vdash_{\varphi a} \{P\}$$

Tel que, il n'y a pas de solution de la forme $\vdash_{\varphi a \phi}$ dans le contexte chimique pour tout φ, ψ dans $\mathcal{X}^*$.

Migration

On rajoute dans la syntaxe des processus une nouvelles primitives pour la migration :

$$P \stackrel{def}{=} ...|go < b, k >$$

avec une nouvelle règle chimique de réduction :

$$\textbf{Go} \; a[D : P|go < b, k >] \vdash_{\varphi} \,||\, \vdash_{\varphi b} \rightarrow \vdash_{\varphi} ||a[D : P|k <>] \vdash_{\varphi b}$$

La location a déplace de sa position courante φa dans l'arbre vers une nouvelle position $\varphi b a$ juste sous la location nommée b. La solution destination $\vdash_{\varphi b}$ est définie par son nom b. Une fois a est arrivée, la continuation $k <>$ peut déclencher d'autres calculs.

Pannes

Join-calcul fournit également un modèle élémentaire de pannes. L'arrêt brutal d'une machine peut causer la terminaison des emplacements qui y résident, de manière irréversible. De manière plus générale, chaque emplacement peut s'arrêter, entraînant tous ses sous-emplacements. La terminaison d'un emplacement est détectable à partir d'autres emplacements qui continuent à fonctionner, ce qui permet de programmer explicitement la résistance aux pannes si besoin.

2.5 Calcul des Ambients

Nous décrivons dans cette section un modèle pour la programmation mobile et distribuée : le calcul des ambients[Car97][CG98][CG99]. C'est un modèle de calcul de processus utilisant la communication de noms, dans l'esprit de π-calcul.

Nous commençons par définir la syntaxe et la sémantique du calcul des ambients sans communication, dans la deuxième section on introduit la notion de communication dans ce calcul.

2.5.1 Présentation

Ambients

Un ambient ce caractérise principalement par le suivant[CG98] :

- Un ambient est une place bornée où se passe le calcul. La propriété intéressante ici c'est l'existence de limite autour d'un ambient. Les bornes déterminent ce qui appartient et ce qui n'appartient pas à l'ambient. Exemple d'ambient dans ce sens, une page web (limitée par un fichier).
- Un ambient peut déplacer dans son ensemble. Si nous rebranchons un portable à un réseau différent, tous les espaces d'adressage et systèmes de fichier dans le portable, déplacent en conséquence et automatiquement.

2.5.2 Mobilité

Primitifs de mobilité

Le calcul décrit la migration de processus réside dans des domaines administratifs. La syntaxe de calcul est décrite par la figure 2.16, page 49 ; un processus ambient P est soit le processus inerte, soit une composition parallèle $P|P$, soit un processus répliqué $!P$, soit un ambient $n[P]$ nommé n, soit un processus $va.P$ avec un nom local a, soit un processus $M.P$ gardé par la capacité M.

Noms libre

$$\mathrm{fn}((vn)P) \stackrel{\Delta}{=} \mathrm{fn}(P) - \{n\} \qquad \mathrm{fn}(in\ n) \stackrel{\Delta}{=} \{n\}$$
$$\mathrm{fn}(0) \stackrel{\Delta}{=} \phi \qquad \mathrm{fn}(out\ n) \stackrel{\Delta}{=} \{n\}$$
$$\mathrm{fn}(P|Q) \stackrel{\Delta}{=} \mathrm{fn}(P) \cup \mathrm{fn}(Q) \qquad \mathrm{fn}(open\ n) \stackrel{\Delta}{=} \{n\}$$
$$\mathrm{fn}(!P) \stackrel{\Delta}{=} \mathrm{fn}(P)$$
$$\mathrm{fn}(n[P]) \stackrel{\Delta}{=} \{n\} \cup \mathrm{fn}(P)$$
$$\mathrm{fn}(M.P) \stackrel{\Delta}{=} \mathrm{fn}(M) \cup \mathrm{fn}(P)$$

Un ambiant est noté par $n[P]$, où n c'est le nom de l'ambient, et P un processus en cours d'exécution à l'intérieur de l'ambient, il peut être une composition parallèle de processus.

N		**Noms**
$P, Q ::=$		**Processus**
	$(vn)P$	*Restriction*
	$\mid\ !\ P$	*Réplication*
	$\mid\ P\|Q$	*Composition parallèle*
	$\mid\ \mathbf{0}$	*Processus inerte*
	$\mid\ n[P]$	*Ambient*
	$\mid\ M.P$	*Action*
$M ::=$		**Capacité**
	in n	*Migration entrante*
	out n	*Migration sortante*
	open n	*dissolution d'ambient*

FIG. 2.16 – *Syntaxe du calcul des ambient sans communication*

En général, les ambients sont organisés en structure d'arbre. La racine de l'arbre représente le réseau. Les sous-ambients, plus ou moins profonds dans l'arbre, sont des entités mobiles logiques ou physiques. Les migrations sont locales et dépendent de la structure locale de l'arbre. Un ambient peut migrer hors de son *père*, ou entrer dans un de ses *frères*. Un ambient peut aussi dissoudre un ambient fils. Pour qu'un ambient atteigne un ambient distant, il doit connaître la structure de l'arbre afin d'effectuer des migrations élémentaires le rapprochant de sa destination.

Le processus $M.P$, exécute la capacité M, puis continu le processus P. Le processus P, ne peut être exécuté jusqu'à ce que l'action soit exécutée.

Capacité d'entrée :

Une capacité d'entrée, *in* m, peut être utilisée dans l'action *in* $m.P$, qui représente une entrée à un ambient nommé m. La règle de réduction est :

$$n[in\ m.P|Q] \mid m[R] \rightarrow m[n[P|Q]|R]$$

Capacité de sortie :

Une capacité de sortie, *out* m, peut être utilisée dans l'action *out* $m.P$, qui représente une sortie de l'ambient nommé m. La règle de réduction est :

$$m[n[out\ m.P|Q]|R] \rightarrow n[P|Q]|m[R]$$

Capacité d'ouverture :

Une capacité d'ouverture, *open* m, peut être utilisée dans l'action *open* $m.P$, L'action fournit une façon de dissoudre la limite d'un ambient nommé m, situé au même niveau que *open*, la règle de réduction est :

$$Open\ m.P|m[Q] \rightarrow P|Q$$

2.5.3 Sémantique opérationnelle

La sémantique du calcul est basée sur une relation de congruence structurelle $\equiv$, entre les processus, et une relation de réduction $\rightarrow$.

La congruence structurelle $\equiv$ est la plus petite relation d'équivalence sur les processus qui satisfait les équations et règles suivantes :

Congruence structurelle

$P \equiv P$	$P\|Q \equiv Q\|P$
$Q \equiv P \Longrightarrow P \equiv Q$	$(P\|Q)\|R \equiv P\|(Q\|R)$
$P \equiv Q, Q \equiv R \Longrightarrow P \equiv R$	$!P \equiv P\|!P$
	$(vn)(vm)P \equiv (vm)(vn)P$

$P \equiv Q \Longrightarrow (vn)P \equiv (vn)Q$	$(vn)(P\|Q) \equiv P\|(vn)Q$ si $n \notin \text{fn}(P)$
$P \equiv Q \Longrightarrow P\|R \equiv Q\|R$	$(vn)(m[P]) \equiv m[(vn)P]$ si $n \neq m$
$P \equiv Q \Longrightarrow !P \equiv !Q$	$P\|0 \equiv P$
$P \equiv Q \Longrightarrow n[P] \equiv n[Q]$	$(vn)0 \equiv 0$
$P \equiv Q \Longrightarrow M.P \equiv M.Q$	$!0 \equiv 0$

A noté que les termes suivants sont différents :

$!(vn)P \equiv\ (vn)!P$ La réplication crée de nouveaux noms

$n[P]|n[Q] \equiv n[P|Q]$ Les différents ambients n, ont des identités séparés.

Réduction

La réduction d'ambient $\rightarrow$, est la plus petite relation sur les processus qui satisfait les règles suivantes :

$n[in\ m.P\|Q]\|m[R] \rightarrow m[n[P\|Q]\|R]$	$P \rightarrow Q \Longrightarrow P\|R \rightarrow Q\|R$
$m[n[out\ m.P\|Q]\|R] \rightarrow n[P\|Q]\|m[R]$	$P \rightarrow Q \Longrightarrow (vn)P \rightarrow (vn)Q$
$open\ n.P\|n[Q] \rightarrow P\|Q$	$P \rightarrow Q \Longrightarrow n[P] \rightarrow n[Q]$
$P\prime \equiv P, P \rightarrow Q, Q \equiv Q\prime \Longrightarrow P\prime \rightarrow Q\prime$	

2.5.4 Communication

Primitives de communication

On commence par la syntaxe, les primitives de la mobilité sont les même de la section précédente, mais l'addition des variables de communication change quelques termes (Voir la figure 2.17, page 51).

Noté que $P\{x \longleftarrow M\}$ c'est la substitution de la capacité M pour chaque occurrence de la variable x dans le processus P. De la même façon pour $M\{x \longleftarrow M\prime\}$.

N		**Noms**
$P, Q ::=$		**Processus**
	$(vn)P$	*Restriction*
	$\mid\ !\ P$	*Réplication*
	$\mid\ P \mid Q$	*Composition parallèle*
	$\mid\ \mathbf{0}$	*Processus inerte*
	$\mid\ M[P]$	*Ambient*
	$\mid\ M.P$	*Action*
	$\mid (x).P$	*Réception de message*
	$\mid < M >$	*Message asynchrone*

$M ::=$		**Capacité**
	x	
	n	
	in M	*Migration entrante*
	out M	*Migration sortante*
	open M	*Dissolution d'ambient*
	ε	*Nul*
	$M.M\prime$	*Chemin*

FIG. 2.17 – *Syntaxe du calcul des ambient avec communication*

Valeurs de la communication

Les entités qui peuvent être communiquées sont des noms ou capacités. Dans les situations réalistes, la communication de noms devrait être plutôt rare.

Il devient maintenant utile de combiner des capacités multiples dans des chemins, surtout quand un ou plus de ces capacités sont représentées par des variables. Par exemple, $(outn.inm).P$ Les variables peuvent être remplacées par des noms ou des capacités.

Le mécanisme de communication le plus simple que nous pouvons imaginer, est une communication locale anonyme dans un ambiant :

$$(x).P \mid < M > \rightarrow P\{x \longleftarrow M\}$$

$\text{fn}(M[P]) \triangleq \text{fn}(M) \cup \text{fn}(P)$ $\quad$ $\text{fn}(x) \triangleq \phi$
$\text{fn}((x).P) \triangleq \text{fn}(P)$ $\quad$ $\text{fn}(\ n) \triangleq \{n\}$
$\text{fn}(< M >) \triangleq \text{fn}(M)$ $\quad$ $\text{fn}(x) \triangleq \phi$
$\text{fn}(M.M') \triangleq \text{fn}(M) \cup \text{fn}(M')$

FIG. 2.18 – *Noms libres*

$\text{fv}((vn)P) \triangleq \text{fv}(P)$ $\quad$ $\text{fv}(in\ M) \triangleq \text{fn}(M)$
$\text{fv}(0) \triangleq \phi$ $\quad$ $\text{fv}(out\ M) \triangleq \text{fn}(M)$
$\text{fv}(P|Q) \triangleq \text{fv}(P) \cup \text{fv}(Q)$ $\quad$ $\text{fv}(open\ M) \triangleq \text{fn}(M)$
$\text{fv}(!P) \triangleq \text{fv}(P)$ $\quad$ $\text{fv}(x) \triangleq \{x\}$
$\text{fv}(M[P]) \triangleq \text{fv}(M) \cup \text{fv}(P)$ $\quad$ $\text{fv}(n) \triangleq \phi$
$\text{fv}((x).P) \triangleq \text{fv}(P) - \{x\}$ $\quad$ $\text{fv}(\varepsilon) \triangleq \phi$
$\text{fn}(< M >) \triangleq \text{fn}(M)$ $\quad$ $\text{fv}(M.M') \triangleq \text{fv}(M) \cup \text{fv}(M')$
$\text{fv}(M.P) \triangleq \text{fv}(M) \cup \text{fv}(P)$

FIG. 2.19 – *Variables libres*

Sémantique opérationnelle

Congruence structurelle
$P \equiv Q \Longrightarrow M[P] \equiv M[Q]$
$P \equiv Q \Longrightarrow (x).P \equiv (x).Q$
$\varepsilon.P \equiv P$
$(M.M\prime).P \equiv M.M\prime.P$
La règle de réduction :

$$(x).P| < M > \rightarrow P\{x \longleftarrow M\}$$

2.6 D'autres modèles et langages

2.6.1 Flexible Nets

Flexible Nets (FN)[KC08a][KC10], extension du modèle Réseaux de Petri coloré[KC08b], est un modèle de spécification des systèmes reconfigurable. Dans ce modèle la structure des réseaux est dynamique en cours d'exécution, c'est à dire la reconfiguration du réseau s'exprime par une opération qui permet l'ajout et la suppression des nœuds. Dans FN, les places, les transitions et les arcs sont des objets qui peuvent vus comme marquage des places. Ces objets peuvent être signés : positif ou négatif. Un marquage d'une place par des objets négatifs (les nœuds) va être supprimé.

2.6.2 KLAIM

KLAIM[RGR98][RGR00][RGRV00](*Kernel Language for Agents Interaction and Mobility*) est un langage expérimental conçu pour la programmation d'applications réseau, fondé sur le langage de coordination LINDA[Gen85]. KLAIM combine entre un calcul de processus asynchrones d'ordre supérieur et le langage Linda, avec des primitives pour la distribution des processus, la mobilité et la sécurité. LINDA modélise la communication par mémoire partagée à l'aide d'un espace de tuples où sont déposés et sélectionnés (par *pattern matching*) les paramètres effectifs des messages. KLAIM complète ce langage avec des notions de sites (emplacements physiques), de localités(emplacement logiques), d'agents et de migration. Les opérations de base sont asynchrones et permettent de lire, retirer ou déposer un tuple dans un espace de tuples distant ($read(t)@l$, $in(t)@l$, $out(t)@l$), de déclencher l'exécution à distance d'un processus ($eval(P)@l$) ou de créer un nouveau noeud d'exécution ($newloc(u)$).

L'espace des domaines est plat. Un site est essentiellement une unité de migration, un processus désirant envoyer ou recevoir un tuple sur un site distant doit nommer ce site explicitement. La communication locale se fait comme dans LINDA, de manière anonyme et asynchrone : une valeur est déposée par un processus dans le tuple space local et un autre processus peut le sélectionner par *pattern matching*. Lors d'une interaction distante, le transfert des tuples entre espaces de tuples est implicite. La mobilité est exprimée par les deux primitives ($eval, out$). Dans la première primitve ($eval(P)@l$) migre le processus P à la localité l, le contexte reste fixe. L'autre migration $out(P)@l$ migre le processus P en l, le contexte dans ce cas est mobile.

2.6.3 M-Calcul

M-Calcul[SS03], est un modèle formel de programmation répartie avec mobilité de processus, inspiré à la fois du calcul des ambients et du Join-calcul. M-Calcul autorise la programmation explicite du comportement des localités. Plus précisément, le M-calcul comprend : des localités aux comportements programmables (domaines), des processus d'ordre supérieur, de la mobilité de processus et une forme de liaison dynamique.

M-Cacul, conserve de Join-calcul la facilité à communiquer à distance, la facilité d'implémentation et garde la notion de définitions sous forme de filtre de messages associé à un processus.

Du calcul des ambients, conserve les aspects très locaux de la migration et de la communication, permettant de contrôler les messages et agents franchissant une frontière de localité. Chaque localité du M-calcul, est dotée d'un *contrôleur*, c'est à dire un processus du M-calcul contrôlant l'interaction du contenu du domaine avec le monde extérieur au domaine.

2.6.4 Obliq

Dans Obliq[Car95], les objets sont les principaux abstractions de structuration de calculs distribués. Ce langage est définit autour de l'idée de la transparence réseau. Les objets dans

le langage Obliq sont des collections de champs nommés. La sémantique distribuée d'Obliq est basée sur les notions de sites, des localités, des valeurs et des threads. Sites (espaces d'adressage) contiennent les localités, et les localités contiennent des valeurs. Chaque localité appartient à un site unique. La manipulation des sites n'est pas explicite dans la syntaxe, aussi le langage ne possède pas une primitive explicite pour exprimer la migration.

2.6.5 PICT & NomadicPict

PICT[Pie98], est un langage basé sur le π-calcul asynchrones, qui ne supporte pas la programmation distribuée, ni le code mobile. Quelques primitives fournissent la possibilité d'écrire des objets simples. Aucun mécanisme de concurrence spécifique pour des objets a été mis en œuvre à l'intérieur du langage. PICT est un langage de programmation de haut niveau, dans le style fonctionnel.

NomadicPict[Uny02], extension de PICT, avec de notion de localité (site), agent et migration. Les entités de base sont les processus et les agents. Un site en NomadicPict est une unité de communication : les agents qui s'exécutent dans le même site peuvent communiquer les uns avec les autres. Dans un agent, un processus peut communiquer, par le biais de canaux comme dans le π-calcul. Un agent est une unité de migration. Il n'y a pas de communication à distance dans NomadicPict : chaque communication doit avoir lieu dans un site. Toutefois, un protocole de communication à distance transparent est mis en œuvre dans le langage, en utilisant les migrations. On trouve deux types de migration : migration des processus et migration des messages. Les domaines sont structurés de manière hiérarchique.

2.7 Conclusion et discussion

Dans ce chapitre nous avons exploré plusieurs formalismes qui adaptent les besoins de la programmation répartie et mobile. Nous avons vu que le formalisme CCS était le point de départ, il n'inclut pas la notion de mobilité. Par la suite le π-calcul a été défini pour remédier à ce manque. Depuis plusieurs formalismes ont été définis. Cependant peux d'entre eux ont été formalisés algébriquement. Ces modèles ont hérité plusieurs concepts et théories liés au π-calcul.

Le join-calcul hérite la plupart des propriétés de π-calcul asynchrone. La différence essentielle entre les deux, est que dans le join-calcul on connaît statiquement tous les récepteurs à un nom de canal donné. Le comportement de synchronisation de noms est entièrement déclaré comme des join-patterns, quand les noms sont présentés dans un processus de définition, compilés dans l'ensemble.

Néanmoins tous ces modèles ignorent la dimension temps des applications distribuées. Le chapitre suivant se focalisera sur les modèles temps-réel qui seront, conjointement aux modèles déjà étudié, une source d'inspiration pour la définition d'un modèle plus général.

Chapitre 3

Modèles et calculs temps réel

Sommaire

Le besoin de spécifier formellement des systèmes dont le comportement dépend étroitement du temps n'est pas nouveau. La plupart des protocoles contiennent des mécanismes de temporisation essentiels pour la sûreté de fonctionnement. La retransmission de messages dans un protocole de liaison de données ou de transport en est un exemple évident.

Vu l'importance du problème, la recherche d'une technique de description formelle apte à spécifier ces systèmes s'est intensifiée durant ces dernières années. La plupart des travaux portent sur l'extension de modèles existants et en particulier sur les algèbres de processus. Parmi ces travaux, portent spécifiquement sur l'extension de LOTOS[BL91][Led91][LL97][LL98] [CdO95a][CdS93a][CSLO00][SC03a].

Dans ce chapitre nous explorons deux extensions de LOTOS, le langage RT-LOTOS qui intègre les contraintes temporelles, et le langage D-LOTOS qui prend en charge les durées des actions.

3.1 Extension temporelle de LOTOS

Principales extensions temporelles à LOTOS La problématique de l'expression explicite du temps dans LOTOS a engendré une série d'approches différentes. Nous listons les

principaux modèles ci-dessous :

- TIC-LOTOS (Quemada 1987, UPM - Espagne)
- LOTOS-T (Miguel 1992, UPM - Espagne)
- T-LOTOS et U-LOTOS (Bolognesi 1991, CNUCE - Italie)
- TLOTOS (Leduc 1992, ULG - Belgique)
- Time LOTOS et ET-LOTOS (Leduc, Leonard 1993-94, ULG - Belgique)
- TE-LOTOS (Leduc, Leonard, Quemada, Miguel et al., 1995)
- LOTOS/T (Nakata 1993, ES-Osaka - Japon)
- RT-LOTOS (Courtiat, de Camargo, Saidouni 1993, LAAS - France)
- E-LOTOS (ISO/IEC 15437 :2001)
- D-LOTOS (Saidouni, Courtiat 2003, LIRE - Algérie et LAAS - France)

Certaines de ces approches ont servi de prémisses et de base de réflexion aux approches qui ont suivi. Pour différencier ces approches, les aspects suivants sont considérés :

Choix du domaine temporel

1. Soit *discret* : les grandeurs temporelles sont définies sur les entiers naturels ($\mathbb{Z}$), une progression d'une unité de temps peut alors être transcrite par l'occurrence d'une action spécifique (fréquemment notée `tic`). Ceci facilite la définition formelle, car le modèle rapproche du modèle non temporisé, mais occasionne un fort risque d'explosion combinatoire.

2. Soit *dense* et *dénombrable* : les grandeurs temporelles sont définies sur les rationnels positifs $\mathbb{Q}^+$. Cela semble parfaitement convenir à la grande majorité des cas pratiques d'utilisation d'un langage de spécification formelle de systèmes temps-réel. Le modèle mathématique sous-jacent est par contre, plus complexe à définir et à mettre en œuvre dans le cadre de la vérification.

3. Soit *réel* : quelques rares modèles évoquent la possibilité de définir les grandeurs temporelles dans $\mathbb{R}^+$. Notons qu'alors, presque aucune technique de validation n'a été proposée.

Temporisation des actions Cette temporisation se fait soit par l'ajout d'un opérateur temporel dédié («*opérateur* $< \ldots >$ P»), soit par une extension de l'opérateur de préfixe (préfixe d'un processus par une action : «$g < \ldots >; P$»), soit les deux.

Hypothèses d'urgence des actions Une action est dite *urgente* lorsqu'elle doit se réaliser immédiatement, sans progression possible du temps, dès qu'elle est sensibilisée. Certains modèles présupposent implicitement que toutes les actions sont urgentes. La plupart associent l'urgence aux actions internes (l'action i ou une action intériorisée par hide). Certains modèles introduisent un mot-clé spécifique pour déclarer urgentes des actions internes ou observables. D'autres, enfin, proposent des mécanismes pour relâcher l'urgence des actions internes sous certaines conditions.

Opérateurs additionnels Certains langages proposent des facilités d'écriture pour spécifier des comportements réputés classiques par le biais d'opérateurs de haut niveau, qui toutefois n'introduisent pas véritablement de fonctionnalités nouvelles dans le modèle (en d'autres termes, les comportements spécifiés par ces opérateurs de haut niveau peuvent également être spécifiés au moyen d'opérateurs du modèle de base, mais de manière plus lourde).

Le tableau 3.1, page 58 reprend quelques points de comparaison entre les différentes extensions temporelles à LOTOS.

3.1.1 *Real-Time* LOTOS (RT-LOTOS)

RT-LOTOS [CdS93a][Loh02], extension temporelle de LOTOS qui a été définie en considérant une sémantique d'entrelacement. Nous en évoquerons quelques aspects dans cette section, dans la perspective d'expliquer certains opérateurs temporels qui ont été repris dans D-LOTOS avant d'introduire celui-ci.

L'introduction de RT-LOTOS a été faite dans le but de permettre la description de l'aspect quantitatif (et non pas seulement de l'aspect qualitatif de l'ordonnancement des événements) des instants auxquels les événements se produisent réellement. L'extension temporelle RT-LOTOS reprend les concepts et l'essentiel des opérateurs de LOTOS, et apporte de nouveaux opérateurs permettant d'exprimer des contraintes temporelles quantifiées.

L'occurrence des actions peut être contrainte de la manière suivante :

- En retardant l'occurrence d'un processus de manière déterministe ;
- En retardant l'occurrence d'un processus de manière non-déterministe ;
- En limitant le temps pendant lequel une action est offerte à son environnement.

RT-LOTOS propose essentiellement trois opérateurs pour décrire, de manière intuitive, l'expression du temps dans le comportement de processus LOTOS.

- L'opérateur de délai (noté `delay`(d) ou Δ^d) permet de retarder un processus d'une certaine quantité de temps d.

	Domaine temporel	**Temporisation**	**Urgence**
TIC-LOTOS	Discret	Préfixe et opérateur	Toute action
LOTOS-T	Discret ou dense	Préfixe	Actions cachées
T-LOTOS	Discret	Opérateur	Mot-clé spécifique
U-LOTOS	Discret	Opérateur	Mot-clé spécifique
TLOTOS	Discret	Opérateur	Relâchée
Time LOTOS	Dense	Opérateur	Actions cachées
ET-LOTOS	Dense	Préfixe et opérateur	Actions cachées
TE-LOTOS	Dense	Préfixe et opérateur	Toute action
LOTOS/T	Discret	Préfixe	Mot-clé spécifique
RT-LOTOS	Dense	Préfixe et opérateur	Actions cachées
E-LOTOS	Dense	Préfixe et opérateur	Actions cachées
D-LOTOS	Dense	Préfixe et opérateur	Actions cachées

TAB. 3.1 – *Comparaison des extensions temporelles à Lotos*

- L'opérateur de latence (noté `latency`(l) ou Ω^l) permet de retarder un processus d'une certaine quantité de temps choisie de manière non-déterministe au sein de l'intervalle de latence $[0, l]$. Notons qu'en fait, cet opérateur porte sur la ou les premières actions du processus auquel il est appliqué. Par ailleurs, il n'a d'effet que s'il porte sur une action interne, l'occurrence d'une action observable étant, en tout état de cause, soumise à la date de l'offre faite par l'environnement. En d'autres termes, l'opérateur de latence permet de relâcher la contrainte d'urgence d'une action interne. Il permet d'introduire de manière générale le non-déterminisme temporel.
- L'opérateur de restriction temporelle (noté $g\{t\}$) limite le temps pendant lequel une action observable g peut être offerte à son environnement. Le délai d'expiration commence à écouler à partir du moment où l'action est offerte.

Présentation informelle des opérateurs de RT-LOTOS Considérons par exemple le processus $Q = delay(d)latency(l)g\{t\}; P$ qui combine trois opérateurs. La figure suivante représente sur une ligne temporelle les contraintes spécifiées : dans l'intervalle $[0, d]$, aucune action n'est offerte ; à une date choisie dans l'intervalle $[d, d+l]$, l'action g est offerte ; l'action g est offerte jusqu'à la date $d + t$, après laquelle elle n'est plus offerte à son environnement.

Syntaxe formelle et sémantique opérationnelle de Basic RT-LOTOS La syntaxe de Basic RT-LOTOS est donnée par la figure 3.1 .

Les notations suivantes seront utilisées lors de l'expression des différentes règles d'inférence de la sémantique opérationnelle :

- $P \xrightarrow{a}$ signifie que $\exists P'$ tel que $P \xrightarrow{a} P'$.
- $P \xnrightarrow{a}$ signifie que le processus P ne peut pas réaliser l'action a.
- $P \xrightarrow{t} P'$ signifie que le processus P ne peut exécuter aucune action pendant une période de t unités de temps et se comporte ensuite comme le processus P'.

Afin de traiter les actions urgentes et non urgentes dans le modèle sémantique, deux actions sémantiques, notées g_w et g_s, sont associées à chaque action $g \in \mathcal{G}$ du modèle syntaxique :

- g_s, appelée action g forte (ou g *strong*),
- g_w, appelée action g faible (ou g *weak*).

Ces actions sémantiques présentent les caractéristiques d'urgence suivantes :

1. g_s et g_w ($g \in \mathcal{G}$) ne sont pas urgentes par définition dès lors que g est observable
2. i_s est urgente, de même que g_s ($g \in \mathcal{G}$) lorsqu'elle est cachée, ainsi que δ_s lorsqu'elle apparaît à droite de l'opérateur $\gg$
3. i_w n'est pas urgente, de même que g_w ($g \in \mathcal{G}$) lorsqu'elle est cachée, ainsi que δ_w lorsqu'elle apparaît à droite de l'opérateur $\gg$

L'*urgence* en RT-LOTOS est définie formellement par l'assertion suivante :

$$P \xrightarrow{i_s} \Rightarrow \forall t \neq 0,\ P \xnrightarrow{t}$$

La sémantique opérationnelle d'entrelacement de Basic RT-LOTOS est définie dans le tableau suivant :

$$\begin{aligned} P ::= {} & \texttt{stop} \mid \texttt{exit} \mid X[L] \mid i; P \mid g; P \\ & \mid P \ [] \ P \mid P \ |[L]| \ P \mid \texttt{hide}\ L\ \texttt{in}\ P \\ & \mid P \gg P \mid P \ [> P \\ & \mid \Delta^u P \mid \Omega^u P \\ & \mid i\{u\}; P \mid g\{u\}; P \end{aligned}$$

FIG. 3.1 – *Syntaxe de Basic RT-LOTOS*

3.2 Le langage D-LOTOS

Dans cette section nous introduisons une approche intégrant à la fois contraintes temporelles et durées des actions dans un langage très proche syntaxiquement de ET-LOTOS[Led91] [LL97] [LL98], mais pour lequel les auteurs de[SC03a][SC03b] ont défini une sémantique de

vrai parallélisme, appelée sémantique temporelle de maximalité. Pour mettre en évidence l'incompatibilité de la sémantique d'entrelacement avec l'attribution de durées aux actions, considérons l'exemple des deux expressions de comportement Basic LOTOS définies par : $E = a; stop|||b; stop$ et $F = a; b; stop[]b; a; stop$. Tant que les durées des actions a et b sont nulles, les deux comportements paraissent identiques, cependant si nous considérons que durée $(a) > 0$ et durée $(b) > 0$, il en découle que dans E l'exécution des actions 'a' et 'b' peut être faite dans un temps égal à max{durée(b), durée(a) }, alors que le temps minimal pour exécuter les deux actions 'a' et 'b' dans F est de durée(a)+durée(b).

3.2.1 Sémantique de maximalité

Dans cette section, nous introduisons la sémantique de maximalité de Basic LOTOS telle qu'elle a été définie en [SC03a]. La syntaxe de Basic LOTOS est définie dans la page 28.

Principe de la sémantique de maximalité

La sémantique d'un système concurrent peut être caractérisée par l'ensemble des états du système et des transitions par lesquelles le système passe d'un état à un autre. Dans l'approche basée sur la maximalité, les transitions sont des événements qui ne représentent que le début de l'exécution des actions. En conséquence, l'exécution concurrente de plusieurs actions devient possible, c'est-à-dire que l'on peut distinguer exécutions séquentielles et exécutions parallèles d'actions.

Etant donné que plusieurs actions qui ont le même nom peuvent s'exécuter en parallèle (auto-concurrence), pour distinguer les exécutions de chacune des actions, un identificateur a été associé à chaque début d'exécution d'action, c'est-à-dire à la transition ou à l'événement associé. Dans un état, un événement est dit maximal s'il correspond au début de l'exécution d'une action qui peut éventuellement être toujours en train de s'exécuter dans cet état là. Associer des noms d'événements maximaux aux états nous conduit à la notion de configuration qui sera formalisée dans la définition 3.2.

Pour illustrer la maximalité et cette notion de configuration, considérons les expressions de comportement E et F introduites précédemment.

Dans l'état initial, aucune action n'a encore été exécutée, donc l'ensemble des événements maximaux est vide, d'où les configurations initiales suivantes associées à E et F : ${}_{\phi}[E]$ et ${}_{\phi}[F]$. En appliquant la sémantique de maximalité, les transitions suivantes sont possibles :

$$ {}_{\phi}[E] \xrightarrow{\phi^{a_x}}_m {}_{\{x\}}[stop] \quad ||| \quad {}_{\phi}[b; stop] \xrightarrow{\phi^{b_x}}_m {}_{\{x\}}[stop] \quad ||| \quad {}_{\{y\}}[stop] $$

x (resp y) étant le nom de l'événement identifiant le début de l'action 'a' (respectivement 'b'). Etant donné que rien ne peut être conclu à propos de la terminaison des deux actions 'a' et 'b' dans la configuration ${}_{\{x\}}[stop]|||{}_{\{y\}}[stop]$, x et y sont alors maximaux dans cette configuration. Notons que x est également maximal dans l'état intermédiaire représenté par

la configuration ${}_{\{x\}}[stop]|||_{\phi}[b; stop]$. Pour la configuration initiale, associée à l'expression de comportement F, la transition suivante est possible :

$$ {}_{\phi}[F] \xrightarrow{\phi^{a_x}}_m {}_{\{x\}}[b; stop] $$

Comme précédemment, x identifie le début de l'action 'a' et il est le nom du seul événement maximal dans la configuration ${}_{\{x\}}[b; stop]$. Il est clair que, au vu de la sémantique de l'opérateur de préfixage, le début de l'exécution de l'action 'b' n'est possible que si l'action 'a' a terminé son exécution. Par conséquent, x ne reste plus maximal lorsque l'action 'b' commence son exécution ; l'unique événement maximal dans la configuration résultante est donc celui identifié par y qui correspond au début de l'exécution de l'action 'b'. L'ensemble des noms des événements maximaux a donc été modifié par la suppression de x et l'ajout de y, ce qui justifie la dérivation suivante :

$$ {}_{\{x\}}[b; stop] \xrightarrow{\{x\}^{a_y}}_m {}_{\{y\}}[stop] $$

La configuration ${}_{\{y\}}[stop]$ est différente de la configuration ${}_{\{x\}}[stop] \;|||\; {}_{\{y\}}[stop]$, car la première ne possède qu'un seul événement maximal (identifié par y), alors que la deuxième en possède deux (identifiés par x et y). La configuration ${}_{\{y\}}[stop]$ est différente de la configuration ${}_{\{x\}}[stop] \;|||\; {}_{\{y\}}[stop]$, car la première ne possède qu'un seul événement maximal (identifié par y), alors que la deuxième en possède deux (identifiés par x et y).

Sémantique de maximalité de Basic LOTOS

Définition 3.1 *L'ensemble des noms des événements est un ensemble dénombrable noté $\mathcal{M}$. Cet ensemble est parcouru par $x, y, ...\ M, N, ...$ dénotent des sous-ensembles finis de $\mathcal{M}$. L'ensemble des atomes de support Act est Atm=$2^{\mathcal{M}}_{fn} \times Act \times \mathcal{M}$, $2^{\mathcal{M}}_{fn}$ étant l'ensemble des parties finies de $\mathcal{M}$. Pour $M \in 2^{\mathcal{M}}_{fn}$, $x \in M$ et $a \in Act$, l'atome (M, a, x) sera noté M^{a_x} . Le choix d'un nom d'événement peut se faire de manière déterministe par l'utilisation de toute fonction get : $2^{\mathcal{M}} - \{\phi\} \longrightarrow \mathcal{M}$ satisfaisant get$(\mathcal{M}) \in \mathcal{M}$ pour tout $M \in 2^{\mathcal{M}} - \{\phi\}$.*

Définition 3.2 *L'ensemble $\mathcal{C}$ des configurations des expressions de comportement de Basic LOTOS est le plus petit ensemble défini par induction comme suit :*

- $\forall E \in \mathcal{B}, \forall M \in 2^{\mathcal{M}}_{fn} \;:\; {}_M[E] \in \mathcal{C}$
- $\forall P \in PN, \forall M \in 2^{\mathcal{M}}_{fn} :\; {}_M[P] \in \mathcal{C}$
- *si $\mathcal{E} \in \mathcal{C}$,alors hide L in $\mathcal{E} \in \mathcal{C}$*
- *si $\mathcal{E} \in \mathcal{C}$,et $F \in \mathcal{B}$ alors $\mathcal{E} \gg F \in \mathcal{C}$*
- *si $\mathcal{E}, \mathcal{F} \in \mathcal{C}$, alors $\mathcal{E}$ op $\mathcal{F} \in \mathcal{C}$* $\qquad op \in \{[]\,, |||, ||, |\,[L]\,|, [>\}$

- *si* $\mathcal{E} \in \mathcal{C}$,*et* $\{a_1, ..., a_n\}, \{b_1, ..., b_n\}$ *deux sous ensembles de* $\mathcal{G}$ *alors* $E[b_1/a_1, ..., b_n/a_n] \in \mathcal{C}$

Etant donné un ensemble $M \in 2^{\mathcal{M}}_{fn}$, $_M[...]$ est appelée opération d'encapsulation.

Définition 3.3 *La fonction* $\psi : \mathcal{C} \longrightarrow 2^{\mathcal{M}}_{fn}$ *, qui détermine l'ensemble des noms des événements dans une configuration, est définie récursivement par :*

$$\begin{array}{lll} \psi\left({}_M\left[E\right]\right) = M & \psi\left(\mathcal{E}\left[\right]\mathcal{F}\right) = \psi\left(\mathcal{E}\right) \cup \psi\left(\mathcal{F}\right) & \psi\left(\mathcal{E}\right|\left[L\right]\left|\mathcal{F}\right) = \psi\left(\mathcal{E}\right) \cup \psi\left(\mathcal{F}\right) \\ \psi\left(\text{hide } L \text{ in } \mathcal{E}\right) = \psi\left(\mathcal{E}\right) & \psi\left(\mathcal{E} \gg \mathcal{F}\right) = \psi\left(\mathcal{E}\right) & \psi\left(\mathcal{E}[> \mathcal{F}\right) = \psi\left(\mathcal{E}\right) \cup \psi\left(\mathcal{F}\right) \\ & \psi\left(\mathcal{E}\left[b_1/a_1, ..., b_n/a_n\right]\right) = \psi\left(\mathcal{E}\right) & \end{array}$$

Définition 3.4 *La relation de transition de maximalité* $\longrightarrow \subseteq \mathcal{C} \times Atm \times \mathcal{C}$ *est définie comme étant la plus petite relation satisfaisant les règles suivantes :*

1. $\dfrac{}{{}_M[exit] \overset{M^{\delta_x}}{\rightarrow} {}_{\{x\}}[stop]} x = get(\mathcal{M})$
2. $\dfrac{}{{}_M[a;E] \overset{M^{a_x}}{\rightarrow} {}_{\{x\}}[E]}\ x = get(\mathcal{M})$
3. (a) $\dfrac{\mathcal{E} \overset{M^{a_x}}{\rightarrow} \mathcal{E}'}{\mathcal{F}[]\mathcal{E} \overset{M^{a_x}}{\rightarrow} \mathcal{E}' \quad \mathcal{E}[]\mathcal{F} \overset{M^{a_x}}{\rightarrow} \mathcal{E}'}$
4. (a) i. $\dfrac{\mathcal{E} \overset{M^{a_x}}{\rightarrow} \mathcal{E}' \quad a \notin L \cup \{\delta\}}{\mathcal{E}|[L]|\mathcal{F} \overset{M^{a_x}}{\rightarrow} \mathcal{E}'[y/x] \ |[L]| \ \mathcal{F} \setminus M} y = get(\mathcal{M} - ((\psi(\mathcal{E}) \cup \psi(\mathcal{F})) - M)$

   ii. $\dfrac{\mathcal{E} \overset{M^{a_x}}{\rightarrow} \mathcal{E}' \quad a \notin L \cup \{\delta\}}{\mathcal{F}|[L]|\mathcal{E} \overset{M^{a_x}}{\rightarrow} \mathcal{F} \setminus M \ |[L]| \ \mathcal{E}'[y/x]} y = get(\mathcal{M} - ((\psi(\mathcal{E}) \cup \psi(\mathcal{F})) - M)$

   (b) $\dfrac{\mathcal{E} \overset{M^{a_x}}{\rightarrow} \mathcal{E}' \quad \mathcal{F} \overset{M^{a_y}}{\rightarrow} \mathcal{F} \quad a \in L \cup \{\delta\}}{\mathcal{E}|[L]|\mathcal{F} \overset{M \cup N^{a_x}}{\rightarrow} \mathcal{E}'[z/x] \setminus N \ \ |[L]| \ \mathcal{F}'[z/y] \setminus M} z = get(\mathcal{M} - ((\psi(\mathcal{E}) \cup \psi(\mathcal{F})) - (M \cup N)))$
5. (a) $\dfrac{\mathcal{E} \overset{M^{a_x}}{\rightarrow} \mathcal{E}' \quad a \notin L}{hide\ L\ in\ \mathcal{E} \overset{M^{a_x}}{\rightarrow} hide\ L\ in\ \mathcal{E}'}$

   (b) $\dfrac{\mathcal{E} \overset{M^{a_x}}{\rightarrow_m} \mathcal{E}' \quad a \in L}{hide\ L\ in\ \mathcal{E} \overset{M^{i_x}}{\rightarrow} hide\ L\ in\ \mathcal{E}'}$
6. (a) $\dfrac{\mathcal{E} \overset{M^{a_x}}{\rightarrow_m} \mathcal{E}' \quad a \neq \delta}{\mathcal{E} \gg \mathcal{F} \overset{M^{a_x}}{\rightarrow} \mathcal{E}' \gg \mathcal{F}}$

   (b) $\dfrac{\mathcal{E} \overset{M^{\delta_x}}{\rightarrow} \mathcal{E}'}{\mathcal{E} \gg F \overset{M^{i_x}}{\rightarrow} {}_{\{x\}}[F]}$
7. (a) $\dfrac{\mathcal{E} \overset{M^{a_x}}{\rightarrow} \mathcal{E}' \quad a \neq \delta}{\mathcal{E}[> \mathcal{F} \overset{M^{a_y}}{\rightarrow} \mathcal{E}'[y/x][> \mathcal{F} \setminus M} y = get(\mathcal{M} - (\psi(\mathcal{E}) \cup \psi(\mathcal{F})) - M)$

   (b) $\dfrac{\mathcal{E} \overset{M^{\delta_x}}{\rightarrow} \mathcal{E}'}{\mathcal{E}[> \mathcal{F} \overset{M^{\delta_y}}{\rightarrow} \mathcal{E}'[y/x][>_{\psi(\mathcal{F})-M}[stop]} y = get(\mathcal{M} - (\psi(\mathcal{E}) \cup \psi(\mathcal{F})) - M)$

(c) $\dfrac{\mathcal{F} \overset{M^{\delta x}}{\rightarrow} \mathcal{F}'}{\mathcal{E}[>\mathcal{F} \overset{M^{\delta y}}{\rightarrow}_{\psi(\mathcal{E})-M}[stop]\ [>\mathcal{F}'[y/x]} y = get(\mathcal{M} - (\psi(\mathcal{E}) \cup \psi(\mathcal{F})) - M)$

8. (a) $\dfrac{\mathcal{E} \overset{M^{ax}}{\rightarrow} \mathcal{E}' \quad a \notin \{a_1,...,a_n\}}{\mathcal{E}[b_1/a_1,...,b_n/b_n] \overset{M^{ax}}{\rightarrow} \mathcal{E}'[b_1/a_1,...,b_n/b_n]}$

(b) $\dfrac{\mathcal{E} \overset{M^{a}x}{\longrightarrow} \mathcal{E}' \quad a=a_i\ (1 \leq i \leq n)}{\mathcal{E}[b_1/a_1,...,b_n/a_n] \overset{M^{bi}x}{\longrightarrow} \mathcal{E}'[b_1/a_1,...,b_n/a_n]}$

9. $\dfrac{P:=E \qquad {}_M[E] \overset{M^{a}x}{\longrightarrow} \mathcal{F}}{{}_M[P] \overset{M^{a}x}{\longrightarrow} \mathcal{F}}$

3.2.2 Introduction des durées et des contraintes temporelles

Soit $\mathcal{D}$ un ensemble dénombrable, les éléments de $\mathcal{D}$ désignent des valeurs temporelles. Soit $\mathcal{T}$ l'ensemble de toutes les fonctions $\tau : Act \longrightarrow \mathcal{D}$ telles que $\tau(i) = t(\delta) = 0$. τ_0 est la fonction constante définie par $\tau_0(a) = 0$ pour tout $a \in Act$.

La fonction de durée τ étant fixée, considérons l'expression de comportement $G = a; b; stop$. Dans l'état initial, aucune action n'est en cours d'exécution et la configuration associée est donc ${}_\phi[a; b; stop]$; à partir de cet état, la transition ${}_\phi[a; b; stop] \overset{\phi^{ax}}{\longrightarrow}_{\{x\}} [b; stop]$ est possible. L'état résultant interprète le fait que l'action a est potentiellement en cours d'exécution. Selon la sémantique de maximalité, nous n'avons pas le moyen de déterminer si l'action a a terminé ou pas son exécution, sauf dans le cas où l'action b a débuté son exécution (le début de b dépend de la fin de a) ; ainsi, si b a débuté son exécution, nous pouvons déduire que a a terminé de s'exécuter. Nous pouvons ainsi constater que les durées d'action sont présentes de manière intrinsèque mais implicite dans l'approche de maximalité ; leur prise en compte de manière explicite va nous permettre de raisonner sur des propriétés quantitatives du comportement d'un système.

En prenant en compte la durée de l'action a, nous pouvons accepter la transition suivante : ${}_\phi[a; b; stop] \overset{\phi^{ax}}{\longrightarrow}_{\{x:a:\tau(a)\}} [b; stop]$. La configuration résultante montre que l'action b ne peut débuter son exécution que si une durée égale à $\tau(a)$ s'est écoulée, cette durée ne représentant rien d'autre que le temps nécessaire à l'exécution de l'action a. Nous pouvons bien sûr également considérer les états intermédiaires représentant l'écoulement d'un laps de temps $t \leq \tau(a)$ par ${}_{\{x:a:\tau(a)\}}[b; stop] \overset{t}{\longrightarrow}_{\{x:a:\tau(a)-t\}} [b; stop]$; de telles configurations seront appelées par la suite configurations temporelles, ce qui nous amène à constater qu'une configuration générée par la sémantique de maximalité représente en fait une classe de configurations temporelles. La prise en compte explicite des durées d'action dans les algèbres de processus ne permet pas seule de spécifier des systèmes temps réel. Pour combler ce manque, les opérateurs classiques de délai similaires à ceux introduits dans des extensions temporelles de LOTOS sont utilisés, telles que ET-LOTOS ou RT-LOTOS, la sémantique de ces opérateurs étant bien entendu exprimée dans le contexte de maximalité. Du fait que les actions ne sont pas atomiques, les contraintes temporelles concernent dans ce contexte le début d'exécution des actions et non pas l'exécution complète des actions. Le langage ainsi défini est appelé D-LOTOS, pour LOTOS avec durées d'actions.

La syntaxe de D-LOTOS est définie comme suit :

$$E ::= \quad stop \mid exit\{d\} \mid \Delta^d E \mid X[L] \mid g@t[SP]; E \mid i@t\{d\}; E \mid \\ E[]E \mid E|[L]|E \mid \ hide\ L\ in\ E \mid E \gg E \mid E\,[> E$$

Soient a une action (observable ou interne), E une expression de comportement et $d \in \mathcal{D}$ une valeur dans le domaine temporel. Intuitivement $a\{d\}$ signifie que l'action a doit commencer son exécution dans l'intervalle temporel $[0, d]$. $\Delta^d E$ signifie qu'aucune évolution de E n'est permise avant l'écoulement d'un délai égal à d. Dans $g@t[SP]; E$ (resp. $i@t\{d\}; E$) t est une variable temporelle mémorisant le temps écoulé depuis la sensibilisation de l'action g (resp. i)et qui sera substituée par zéro lorsque cette action termine son exécution.

Définition 3.5 *L'ensemble $\mathcal{C}_t$ des configurations temporelles est donné par :*

- $\forall E \in \mathcal{B}, \forall M \in 2^{\mathcal{M}\times Act\times\mathcal{D}}_{fn} \ : \ {}_M[E] \in \mathcal{C}_t$
- $\forall P \in PN, \forall M \in 2^{\mathcal{M}\times Act\times\mathcal{D}}_{fn} : \ {}_M[P] \in \mathcal{C}_t$
- *si* $\mathcal{E} \in \mathcal{C}_t$ *alors hide* L *in* $\mathcal{E} \in \mathcal{C}_t$
- *si* $\mathcal{E} \in \mathcal{C}_t$ *et* $F \in \mathcal{B}$ *alors* $\mathcal{E} \gg F \in \mathcal{C}_t$
- *si* $\mathcal{E}, \mathcal{F} \in \mathcal{C}_t$, *alors* $\mathcal{E}$ *op* $\mathcal{F} \in \mathcal{C}_t \qquad op \in \{[]\,, |||, ||, |\,[L]\,|, [>\}$
- *si* $\mathcal{E} \in \mathcal{C}_t$ *et* $\{a_1, ..., a_n\}\,, \{b_1, ..., b_n\} \in 2^{\mathcal{G}}_{fn}$ *alors* $E[b_1/a_1, ..., b_n/a_n] \in \mathcal{C}_t$
- $\forall \mathcal{E} \in \mathcal{C}_t, \forall d \in \mathcal{D} : \Delta^d \mathcal{E} \in \mathcal{C}_t$
- ${}_{(x:g:d)}[E(t)] \in \mathcal{C}_t$

Définition 3.6 *L'opérateur d'écoulement de temps (.)d dans une configuration est définie récursivement par :*

$\phi^d = \phi$	$(\mathcal{E}[]\mathcal{F})^d = \mathcal{E}^d[]\mathcal{F}^d$
$(x : a : d')^d = x : a : d' - d$ *tel que* d'-$d = 0$ *si* d>d'	$({}_M[E])^d = {}_{M^d}[E]$
$(M \cup \{x : a : d'\})^d = Md \cup \{(x : a : d')^d\}$	$(\mathcal{E}\|[L]\|\mathcal{F})^d = \mathcal{E}^d\|[L]\|\mathcal{F}^d$
$(hide\ L\ in\ \mathcal{E})^d = hide\ L\ in\ \mathcal{E}^d$	$(\mathcal{E} \gg F)^d = \mathcal{E}^d \gg F$
$(\mathcal{E}[b_1/a_1, ..., b_n/a_n])^d = (\mathcal{E}^d[b_1/a_1, ..., b_n/a_n])$	$(\mathcal{E}[> \mathcal{F})^d = \mathcal{E}^d[> \mathcal{F}^d$
${}_{\{x:g:d'\}}[E(t)]^d = {}_{\{x:g:d'\}^d}[[t + d/t]E(t)]$	

3.2.3 Sémantique opérationnelle structurée de D-LOTOS

La fonction de durée τ étant fixée, la relation de transition temporelle de maximalité entre les configurations temporelles est notée $\rightarrow_\tau \subseteq \mathcal{C}_t \times Atm \cup \mathcal{D} \times \mathcal{C}_t$.

Processus *stop* Considérons la configuration ${}_M[stop]$. A la différence du processus *stop* de LOTOS, cette configuration représente des évolutions potentielles en fonction des actions indexées par l'ensemble M. L'évolution cesse dès que toutes ces actions terminent, ce qui est caractérisé au moyen du prédicat $Wait : 2^{\mathcal{M}\times Act\times\mathcal{D}}_{fn} \longrightarrow \{true, false\}$ défini sur tout $M \in 2^{\mathcal{M}\times Act\times\mathcal{D}}_{fn}$ par : $Wait(M) = \exists\, x : a : d \in M \;\; tel\ que\ d > 0$. Intuitivement $Wait(M) = true$ s'il existe au moins une action référencée dans M qui est en cours d'exécution. Ainsi, tant que $Wait(M) = true$, le passage du temps a un effet sur la configuration ${}_M[stop]$, d'où la règle sémantique ${}_M[stop] \xrightarrow{d}_\tau {}_{M^d}[stop]$ avec M^d donné par la définition 3.6.

Processus *exit* Considérons maintenant la configuration ${}_M[exit]$. La terminaison avec succès ne peut se produire qu'une fois les actions indexées par l'ensemble M ont terminé leur exécution, ce qui est conditionné par la valeur de $Wait(M)$ qui doit être égale à $false$ dans la règle 1. Les règles 2 et 3 expriment le fait que le temps attaché au processus *exit* ne peut commencer à s'écouler que si toutes les actions référencées par M sont terminées. La règle 4 impose que l'occurrence de l'action δ ait lieu dans la période d, dans le cas contraire la terminaison avec succès ne se produira jamais.

1. $\dfrac{\neg Wait(M)}{{}_M[exit\{d'\}] \xrightarrow{M^\delta{}_x}_\tau {}_{\{x:\delta:0\}}[stop]} \qquad x = get(\mathcal{M})$

2. $\dfrac{Wait(M^d)\ or\ (\neg Wait(M^d)\ and\ \forall\varepsilon{>}0.\ Wait(M^{d-\varepsilon})) \qquad d{>}0}{{}_M[exit\{d'\}] \xrightarrow{d}_\tau {}_{M^d}[exit\{d'\}]}$

3. $\dfrac{\neg Wait(M)}{{}_M[exit\{d'{+}d\}] \xrightarrow{d}_\tau {}_M[exit\{d'\}]}$

4. $\dfrac{\neg Wait(M)\ \ and\ \ d'{>}d}{{}_M[exit\{d\}] \xrightarrow{d'}_\tau {}_M[stop]}$

Opérateur de préfixage Les mêmes contraintes sont imposées à l'occurrence d'une action observable préfixant un processus que celles imposées à l'occurrence de l'action δ à partir de la configuration ${}_M[exit\{d\}]$. Avec l'hypothèse que les actions ne sont pas urgentes, nous considérons l'opérateur @ introduit dans ET-LOTOS. L'expression SP dans les règles suivantes représente un prédicat sur l'exécution de l'action g. Les règles 1, 2 et 5 montrent que la prise en compte de l'écoulement du temps dans E commence uniquement lorsque l'action g est sensibilisée ou a commencé son exécution. La règle 3 exprime le fait qu'une fois que l'action g est sensibilisée et que le prédicat SP est vrai à cet instant, l'action g peut commencer son exécution. L'expression de comportement E ne peut évidemment évoluer que si l'action g se termine, ce qui est exprimé par la règle 4. Il est à noter que le préfixage peut être soit par une action observable ou interne avec la condition que l'action interne i et de durée nulle ($\tau(i) = 0$).

1. $\dfrac{Wait(M^d)\ or\ (\neg Wait(M^d)\ and\ \forall\varepsilon{:}0{<}\varepsilon{<}d\ Wait(M^{d-\varepsilon})) \qquad d{>}0}{{}_M[g@t[SP];E] \xrightarrow{d}_\tau {}_{M^d}[g@t[SP];E]}$

2. $\dfrac{\neg Wait(M) \qquad d{>}0}{{}_M[g@t[SP];E] \xrightarrow{d}_\tau {}_M[g@t[[t{+}d/t]SP];[t{+}d/t]E]}$

3. $\dfrac{\neg Wait(M)\ and \vdash [0/t]SP}{{}_M[g@t[SP];E] \xrightarrow{M^g x}_\tau {}_{\{x:g:\tau(g)\}}[E(t)]} \qquad x = get(\mathcal{M})$

4. $\dfrac{\neg Wait(x{:}g{:}d)\ and\ {}_{\{x:g:0\}}[[0/t]E] \xrightarrow{M^a x} \mathcal{E}}{{}_{\{x:g:d\}}[E(t)] \xrightarrow{M^a x}_\tau \mathcal{E}}$

5. $\dfrac{}{{}_{\{x:g:d'+d\}}[E(t)] \xrightarrow{d}_\tau {}_{\{x:g:d'\}}[[t+d/t]E(t)]}$

Les autres opérateurs La sémantique des opérateurs de délai, de choix, de composition parallèle, d'intériorisation, de séquencement, d'interruption, de renommage de portes et d'instanciation de processus sont les même règles sémantique de Basic LOTOS dans lesquelles les configurations sont temporelles et la relation de transition est remplacée par $\rightarrow_\tau$, complétées par les règles suivantes :

$$\frac{}{\Delta^{d'+d}\mathcal{E} \xrightarrow{d}_\tau \Delta^{d'}\mathcal{E}} \qquad \frac{\mathcal{E} \xrightarrow{d} \mathcal{E}'}{\Delta^0\mathcal{E} \xrightarrow{d}_\tau \mathcal{E}'} \qquad \frac{\mathcal{E} \xrightarrow{M^a x}_\tau \mathcal{E}'}{\Delta^0\mathcal{E} \xrightarrow{M^a x}_\tau \mathcal{E}'}$$

$$\frac{\mathcal{E} \xrightarrow{d}_\tau \mathcal{E}' \qquad \mathcal{F} \xrightarrow{d}_\tau \mathcal{F}'}{\mathcal{E}\ []\ \mathcal{F} \xrightarrow{d}_\tau \mathcal{E}'[]\mathcal{F}'} \qquad \frac{\mathcal{E} \xrightarrow{d}_\tau \mathcal{E}' \qquad \mathcal{F} \xrightarrow{d}_\tau \mathcal{F}'}{\mathcal{E}\ |[L]|\ \mathcal{F} \xrightarrow{d}_\tau \mathcal{E}'|[L]|\mathcal{F}'}$$

$$\frac{P{:=}E \qquad {}_M[E] \xrightarrow{d}_\tau \mathcal{F}}{{}_M[P] \xrightarrow{d}_\tau \mathcal{F}}$$

$$\frac{\mathcal{E} \xrightarrow{d} \mathcal{E}' \qquad \forall d'{<}d \quad \mathcal{E}^{d'} \not\xrightarrow{a}_\tau \qquad \forall a{\in}L}{hide\ L\ in\ \mathcal{E} \xrightarrow{d}_\tau hide\ L\ in\ \mathcal{E}'} \qquad \frac{\mathcal{E} \xrightarrow{M^\delta x}_\tau \mathcal{E}'}{\mathcal{E}{>>}F \xrightarrow{M^i x}_\tau {}_{\{x:i:0\}}[F]}$$

$$\frac{\mathcal{E} \xrightarrow{d}_\tau \mathcal{E}' \qquad \mathcal{E} \not\xrightarrow{\delta}_\tau}{\mathcal{E}{>>}F \xrightarrow{d}_\tau \mathcal{E}'{>>}F} \qquad \frac{\mathcal{E} \xrightarrow{d}_\tau \mathcal{E}' \qquad \mathcal{F} \xrightarrow{d}_\tau \mathcal{F}'}{\mathcal{E}[{>}\mathcal{F} \xrightarrow{d}_\tau \mathcal{E}'[{>}\mathcal{F}'}$$

$$\frac{\mathcal{E} \xrightarrow{d}_\tau \mathcal{E}'}{\mathcal{E}[b_1/a_1,...,b_n/a_n] \xrightarrow{d}_\tau \mathcal{E}'[b_1/a_1,...,b_n/a_n]} \qquad \frac{\mathcal{E} \xrightarrow{M^a x}_\tau \mathcal{E}' \qquad a{=}a_i\ (1{\leq}i{\leq}n)}{\mathcal{E}[b_1/a_1,...,b_n/a_n] \xrightarrow{M^{bi} x}_\tau \mathcal{E}'[x{:}bi{:}dbi/x{:}a{:}da][b_1/a_1,...,b_n/a_n]}$$

Ces règles sont similaires à celles introduites dans les algèbres de processus temps-réel.

3.2.4 Relations de bisimulation

Dans cette section nous définissons quelques relations de bisimulation caractérisant les comportements d'applications concurrentes.

Relation de bisimulation temporelle

Définition 3.7 *Soit* $\mathcal{L} = (2^{\mathcal{M}}_{fn} \times Act \times \mathcal{M}) \cup \mathcal{D}$ *et* $\boldsymbol{R} \subseteq \mathcal{C} \times \mathcal{C}$ *une relation binaire entre les configurations temporelles. Soit* $\boldsymbol{F}$ *:* $Rel(\mathcal{C}) \longrightarrow Rel(\mathcal{C})$ *une fonction définie par :* $(E, F) \in \mathbf{F}(\mathbf{R})$ *si :*

1. *(a) Si* $\mathcal{E} \xrightarrow{M^{ax}}_\tau \mathcal{E}'$ *avec* ${}_{M^{ax}} \in 2^{\mathcal{M}}_{fn} \times Act \times \mathcal{M}$*, alors il existe* $\mathcal{F} \xrightarrow{N^{ay}}_\tau \mathcal{F}$ *tel que* $(\mathcal{E}', \mathcal{F}') \in \boldsymbol{R}$

   (b) $\mathcal{E} \xrightarrow{d}_\tau \mathcal{E}'$ *avec* $d \in \mathcal{D}$*, alors il existe* $\mathcal{F} \xrightarrow{d}_\tau \mathcal{F}'$ *tel que* $(\mathcal{E}', \mathcal{F}') \in \mathbf{R}$

2. *(a) Si* $\mathcal{F} \xrightarrow{N^{ay}}_\tau \mathcal{F}\prime$ *avec* ${}_{N^{ay}} \in 2^{\mathcal{M}}_{fn} \times Act \times \mathcal{M}$*, alors il existe* $\mathcal{E} \xrightarrow{M^a_x}_\tau \mathcal{E}'$ *tel que* $(\mathcal{E}', \mathcal{F}) \in \boldsymbol{R}$

   (b) Si $\mathcal{F} \xrightarrow{d}_\tau \mathcal{F}'$ *avec* $d \in \mathcal{D}$*, alors il existe* $\mathcal{E} \xrightarrow{d}_\tau \mathcal{E}'$ *tel que* $(\mathcal{E}', \mathcal{F}') \in \boldsymbol{R}$

***R** est dite une relation de bisimulation temporelle forte ssi **R**⊆**F**(**R**). Si $(\mathcal{E},\mathcal{F}) \in \mathbf{R}$ pour une relation de bisimulation temporelle **R**, alors $\mathcal{E}$ et $\mathcal{F}$ sont dites fortement temporellement bisimilaires, et on note $\mathcal{E} \backsim_t^\tau \mathcal{F}$. Ce qui peut être exprimé par $\backsim_t^\tau = \cup\{\boldsymbol{R} : \boldsymbol{R}$ est une relation de bisimulation temporelle forte$\}$.*

Deux expressions de comportement D-LOTOS E et F sont dites fortement temporellement bisimilaires, noté $E \backsim_t^\tau F$, s'il existe une relation de bisimulation temporelle forte **R** *tel que* $({}_\phi[E],{}_\phi[F]) \in \mathbf{R}$.

Relation de bisimulation temporelle de maximalité

Dans[Saï96] il a été montré que la relation de bisimulation de maximalité est une congruence vis à vis du raffinement d'actions, et par dualité vis à vis de l'association de durées aux actions (conjecture). Ceci a motivé à étendre la relation de bisimulation de maximalité aux configurations temporelles.

Définition 3.8 *Soient* $\mathcal{L} = (2^{\mathcal{M}}_{fn} \times Act \times \mathcal{M}) \cup D$, $\mathfrak{F} \subseteq 2^{\mathcal{M}\times\mathcal{M}}$ *et* $\boldsymbol{R}\subseteq \mathcal{C}\times C\times F$

une relation binaire entre les configurations temporelles. Soit $\boldsymbol{F}^{mt} : Rel(\mathcal{C}) \longrightarrow Rel(\mathcal{C})$ *une fonction définie par :* $(\mathcal{E},\mathcal{F},f) \in \boldsymbol{F}^{mt}(R)$ *si :*

1. $Dom(f) \subseteq \psi(\mathcal{E})$ *et* $Codom(f) \subseteq \psi(\mathcal{F})$,
2. *(a) Si* $\mathcal{E} \xrightarrow{M^{a_x}}_\tau \mathcal{E}'$ *, alors il existe* $\mathcal{F} \xrightarrow{N^{a_y}}_\tau \mathcal{F}'$ *tel que*
   - *i. pour tout* $(u,v) \in f$ *si* $u \notin M$ *alors* $v \notin N$; *et*
   - *ii.* $(\mathcal{E}',\mathcal{F}',f') \in \mathbf{R}$, *avec* $f' = (f(\psi(\mathcal{E}') - \{x\}))(\psi(\mathcal{F}') - \{y\}) \cup \{(x,y)\}$

   (b) Si $\mathcal{E} \xrightarrow{d}_\tau \mathcal{E}'$ *, alors il existe* $\mathcal{F} \xrightarrow{d}_\tau \mathcal{F}'$ *tel que* $(\mathcal{E}',\mathcal{F}',f) \in \mathbf{R}$
3. *(a) Si* $\mathcal{F} \xrightarrow{N^{a_y}}_\tau \mathcal{F}'$ *, alors il existe* $\mathcal{E} \xrightarrow{M^{a_x}}_\tau \mathcal{E}'$ *tel que*
   - *i. pour tout* $(u,v) \in f$ *si* $v \notin N$ *alors* $u \notin M$; *et*
   - *ii.* $(\mathcal{E}',\mathcal{F}',f') \in \mathbf{R}$, *avec* $f' = (f(\psi(\mathcal{E}') - \{x\}))(\psi(\mathcal{F}') - \{y\}) \cup \{(x,y)\}$

   (b) Si $\mathcal{F} \xrightarrow{d}_\tau \mathcal{F}'$ *, alors il existe* $\mathcal{E} \xrightarrow{d}_\tau \mathcal{E}'$ *tel que* $(\mathcal{E}',\mathcal{F}',f) \in \mathbf{R}$

R *est une relation de bisimulation temporelle de maximalité forte ssi* $\mathbf{R} \subseteq \mathbf{F}^{mt}(\mathbf{R})$. *Si* $(\mathcal{E},\mathcal{F},f) \in \boldsymbol{R}$ *pour une certaine relation de bisimulation temporelle de maximalité forte* **R**, *alors les configurations* $\mathcal{E}$ *et* $\mathcal{F}$ *sont dites liées par cette relation, ce qui est noté symboliquement* $\mathcal{E} \backsim_{mt}^\tau \mathcal{F}$. *($f[A)[B]$ désigne la fonction résultante de la restriction du domaine de f à A de son codomaine à B).*

3.3 Limites du langage D-LOTOS

Dans [MS09], nous avons discuté les limites du langage D-LOTOS. La première insuffisance c'est que D-LOTOS est un modèle de spécification statique, il n'offre pas la possibilité de

création dynamique des processus ou de localités, la topologie du réseau de connexion est fixe, malgré que les systèmes distribués sont naturellement dynamique. L'autre insuffisance, concerne le mécanisme de communication élémentaire entre agents. Traditionnellement la communication entre processus est modélisée par la transmission de message d'un processus émetteur vers un processus récepteur. Dans la pratique, la programmation répartie utilise la communication asynchrone, il s'agit d'envois de messages asynchrones d'un point fixe du réseau à un autre. D-LOTOS ne fournit aucune primitive de communication asynchrone. Par ailleurs, il fournit des primitives de communication synchrone. Donc une implémentation du langage D-LOTOS dans un cadre distribué est loin d'être évidente à cause de la synchronisation distribuée présente dans le calcul. Pour que le modèle soit directement implémentable dans un cadre parallèle, il est nécessaire que toute synchronisation présente dans le modèle soit locale, en d'autre terme, interdire les rendez-vous entre agents se trouvant sur deux sites distincts.

Cette analyse souligne que D-LOTOS n'est pas adéquat pour la programmation répartie, même s'il permet d'en étudier certains aspects.

3.4 Conclusion

Dans ce chapitre, nous avons exposé dans la première section l'extension temporelle RT-LOTOS. Ensuite dans la deuxième section, nous avons étudier le modèle de spécification D-LOTOS qui intègre deux concepts à savoir la spécification des contraintes temporelles et la prise en compte des durées d'action. D'un point de vue syntaxique, D-LOTOS restés très proche du formalisme RT-LOTOS, mais d'un point de vue sémantique les auteurs [SC03a], abstrait de l'hypothèse d'atomicité des actions qui est imposée par la sémantique d'entrelacement du parallélisme. Dans ce but, la sémantique de maximalité temporelle a été adoptée pour D-LOTOS.

Désormais, nous avons les outils qui nous permettent de définir un langage général qui remédit aux limites des modèles temps-réel et les modèles dédiés à la mobilité. Ceci fera l'objet de la deuxième partie.

Deuxième partie

Contributions

Chapitre 4

Modèle sémantique distribué pour les systèmes temps-réel distribués et mobiles

Sommaire

Le comportement d'un système réel peut être représenté par un système de transitions sous certaines hypothèses d'abstraction. Ce système de transition est un ensemble de nœuds reliés entre eux. Les transitions sont décorées par des étiquettes représentant les actions exécutées, tandis que les nœuds représentent les états du système. Les systèmes de transitions étiquetées (STEs) sont des modèles très intuitifs lorsqu'on fait abstraction du temps lors de la spécification des systèmes réels. Le besoin de spécifier des systèmes temps-réel a fait naître l'idée d'incorporer le temps dans les STEs, d'où la définition des systèmes de transitions temporisés où les transitions dénotent cette fois-ci soit une exécution d'une action soit un

passage de temps. Le formalisme des automates temporisés (TA's) a été introduit par Rajeev Alur et David Dill dans[AD94][AFH94]. Sa définition fournit un simple moyen pour doter les systèmes de transitions d'un ensemble de contraintes temporelles exprimées à l'aide de variables réelles appelées horloges. Le modèle des automates temporisés est construit en se conformant à l'hypothèse d'atomicité structurelle et temporelle des actions (actions indivisibles et de durées nulles). Le modèle Durational Action Timed Automata (DATA)[BS05] est introduit dans le but de prendre en compte les durées explicites des actions, une extension de cette approche aux systèmes à temps contraint pour pouvoir prendre en considération les contraintes temporelles et l'urgence des actions, ce modèle est appelé DATA*[SB06].

L'objectif de ce chapitre est la définition de deux modèles sémantiques, le premier modèle permet de prendre en charge l'aspect distribué des systèmes répartis temps réel, et le deuxième modèle étend le premier par l'aspect mobilité. Dans une autre section nous avons proposé un algorithme de transformation des DATA* en automate temporisé.

4.1 Automates Temporisés avec Durées d'Actions (DATA*)

Dans cette section, nous décrivons une méthode permettant la prise en compte de la non-atomicité temporelle et structurelle des actions dans les automates temporisés, grâce au modèle des DATA. En général, les systèmes à temps contraint ne peuvent être complètement spécifiés si l'on ne considère pas des notions comme l'urgence, les délais, les contraintes, etc. Pour prendre en compte ces concepts, le modèle des DATA*'s a été défini.

Soit H, parcouru par $x, y...$ un ensemble d'horloges avec des valeurs non négatives (dans le domaine temps $\mathbf{T}$, comme $\mathbf{Q}^+$ ou $\mathbf{R}^+$). L'ensemble $\Phi_t(H)$ des contraintes temporelles γ sur H définis par la syntaxe $\gamma ::= x \sim t$, où x est une horloge dans H, $\sim \in \{=, <, >, \leq, \geq\}$ et $t \in \mathbf{Q}^+$.

Définition 4.1 *Un DATA* A est un quintuplet (S, L_S, s_0, H, T) tel que :*

1. S est un ensemble fini d'états,
2. $L_S : S \rightarrow 2^{\Phi_t(H)}_{fn}$ est une fonction qui fait correspondre à chaque état s l'ensemble F des conditions de terminaison des actions potentiellement en exécution dans s,
3. $s_0 \in S$ est l'état initial,
4. H est un ensemble fini d'horloges, et
5. $T \subseteq S \times 2^{\Phi_t(H)}_{fn} \times 2^{\Phi_t(H)}_{fn} \times Act \times H \times S$ est l'ensemble des transitions. Une transition (s, G, D, a, x, s') représente le passage de l'état s à l'état s', en lançant l'exécution de l'action a et en réinitialisant l'horloge x. G est la contrainte correspondante, qui doit être satisfaite pour tirer cette transition. D est l'échéance correspondante qui exige, au moment de son expiration, que l'action a doit être tirée. (s, G, D, a, x, s') peut être écrit $s \overset{G,D,\alpha,x}{\longrightarrow} s'$.

La sémantique d'un DATA* $A = (S, L_S, s0, H, T)$ est définie en y associant un système de transitions S_A sur $Act \cup \mathbf{T}$. Un état de S_A (ou configuration) est une paire $< s, v >$ tels que s est un état de A et v est une valuation de H. Une configuration $< s_0, v_0 >$ est initial si s_0

est l'état initial de A et $\forall x \in H$, $v_0(x) = 0$. Deux types de transitions entre les configurations de S_A sont possible, qui correspondent respectivement au passage du temps ou le lancement d'une transition à partir de A.

Exemple de DATA* est donné par la figure 4.1

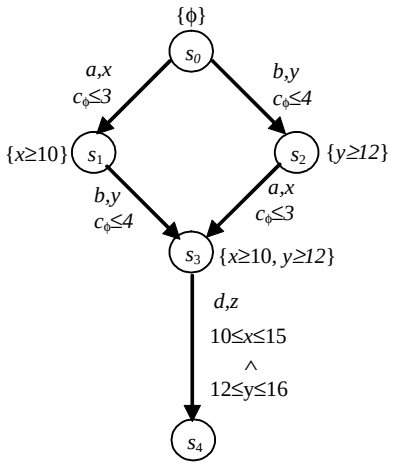

FIG. 4.1 – Exemple de DATA*

4.1.1 Discussion et motivations

Notre sujet d'étude est les systèmes distribués, c'est à dire les systèmes représentés par une collection de domaines ou localités autonomes qui sont connectés à l'aide d'un réseau de communication. La spécification de tels systèmes sera développée autour de la notion de distribution du calcul, plus précisément, l'existence de domaines. Un domaine peut être défini comme étant une région regroupant un ensemble de processus, les calculs de processus traitent cette caractéristique d'une façon implicite ou explicite.

Notre but est de développer le modèle nécessaire pour pouvoir vérifier des propriétés comportementales d'applications distribuées, notamment d'applications communicantes. Pour cela, il est indispensable de concevoir le modèle comportemental général d'une application avant la vérification. Dans ce chapitre, il est donc question de la définition et la génération de modèles comportementaux des systèmes distribués communicantes.

Dans[MS09], nous avons défini un modèle qui distingue entre deux types de calculs, local et global. Pour le calcul local où en peux synchroniser les communications entre les processus, la description peut se faire avec les DATA*, elles suffisent pour décrire un comportement contraint dans le temps, de ce fait chaque localité est décrite par un DATA* (état, action, contraintes et durée d'action), La sémantique de maximalité sous-jacente explicite les durées des actions présentent dans le calcul. Cependant le calcul global représente une tache ardu que les DATA* ne peuvent pas représenter. Nous nous sommes inspirés des deux modèles ;

celui des automates communicants et celui des DATA* pour définir un modèle que nous appelons C-DATA[MSK11] pour (Communicating Durational Action Timed Automata) que nous allons décrire dans la suite de ce chapitre.

4.2 Communicating Durational Action Timed Automata (C-DATA)

Dans cette section, nous introduisons notre modèle sémantique[MSK11], permettant la prise en compte de tout les aspects déjà présents dans le modèle des DATA*s à savoir la non-atomicité temporelle et structurelle des actions, l'urgence des actions, les délais et les contraintes temporelles. En plus notre modèle, dans une première version, prend en charge l'aspect distribué du calcul.

Dans notre modèle chaque localité est représenter par un DATA, le système globale est représenté par l'ensemble des DATAs locaux, qui communiquent entre-eux par échange de messages via des canaux de communications.

4.2.1 Définitions préliminaires

Définition 4.2 *(actions) : Les actions dans le système global sont :*

- L'ensemble des actions de communication entre les localités : L'émission et la réception des messages via les canaux de communication $Act_{com} ::= a!m \mid a?x \mid \tau$ (action d'émission ou de production, action de réception et l'action silencieuse).
- L'ensemble $Act = \mathcal{G} \cup \{i, \delta\}$.

Intuitivement, le terme $a!m$ dénote l'émission du message m sur le canal a, le terme $a?x$ dénote la réception sur le canal a d'une valeur qui est affectée à la variable x, τ désigne l'action interne qui n'engendre aucune communication.

Définition 4.3 *(Localités et canaux de communication) : L'ensemble $\mathcal{L}$ parcouru par l, désigne l'ensemble des localités. ϑ un ensemble infini de canaux définis par les utilisateurs. Cet ensemble est parcouru par a, b, etc. Les canaux sont utilisés pour l'échange de messages entre les localités.*

Définition 4.4 *Un Communicating DATA (C-DATA) est un septuple $A(S, L_S, s_0, \vartheta, H, \Pi, T_D)$ tel que :*

- S est un ensemble fini d'états,
- $L_S : S \rightarrow 2^{\Phi_t(H)}_{fn}$ est une fonction qui fait correspondre à chaque état s l'ensemble F des conditions de terminaison des actions potentiellement en exécution dans s,

- $s_0 \in S$ est l'état initial,
- ϑ l'esemble de l'alphabet des canaux de communication sur lesquels les messages transitent entre les sous-systèmes (DATA),
- H est un ensemble fini d'horloges,
- $\Pi = Act_{com} \cup Act$, est l'esemble des actions locales et de communications,
- $T_D \subseteq S \times 2^{\Phi_t(H)}_{fn} \times 2^{\Phi_t(H)}_{fn} \times \Pi \times H \times S$ est l'ensemble des transitions.

Dans ce modèle une transition $(s, G, D, \alpha/(a(!/?)v)/\tau, z, s\prime)$ représente le passage de l'état s à l'état s', en lançant l'exécution de l'action $\alpha \in Act$ ou action de communication (émission ou réception) ou synchronisation dans une communication (action silencieuse) et en réinitialisant l'horloge z. G est la contrainte correspondante, qui doit être satisfaite pour tirer cette transition. D est l'échéance correspondante qui exige, au moment de son expiration, que l'action α doit être tirée. $(s, G, D, \alpha/(a(!/?)v)/\tau, z, s\prime)$ peut être écrite $s \xrightarrow{G,D,\alpha/(a(!/?)v)/i),z} s'$.

Définition 4.5 ***(Système)*** *Un système $S = (A_1, \ldots, A_n)$, est un C-DATA, tel que chaque $A_i = (S_i, L_{S_i}, s_{0_i}, \vartheta, H_i, \Pi_i, T_{iD})$ $(1 \leq i \leq n)$ est un $DATA$.*

Définition 4.6 *On se donne les notation suivantes :*

Etats *: $GS(S) = (s_1, v_1) \times \ldots \times (s_n, v_n) \times (\vartheta^*)^p$, est l'ensemble des états.*

Etat initial : *L'état initial de S est :$q_0 = ((s_{01}, 0), \ldots, (s_{0n}, 0) : \epsilon_1, \ldots, \epsilon_p)$ tel que ϵ est le mot vide sur l'alphabet des canaux ϑ.*

Etats de système *: Soit $S = (A_1, \ldots, A_n)$ un système de n $DATA$, tel que chaque $A_i = (S_i, L_{S_i}, s_{0_i}, \vartheta, H_i, \Pi_i, T_{iD})$ $(1 \leq i \leq n)$. L'état global de S est défini par l'état de chaque sous-système A_i et l'état de chaque canal, un état de S est un élément de $(s_1, v_1) \times \ldots \times (s_n, v_n) \times (\vartheta^*)^p$ tel que $v_i(h)$ sont les valuations sur H.*

4.2.2 Construction opérationnelle des C-DATA's :

Définition 4.7 *Soit $S = (A_1, \ldots A_n)$ un système de n C-DATA, tel que chaque $A_i = (S_i, L_{S_i}, s_{0_i}, \vartheta, H_i, \Pi_i, T_{iD})$ $(1 \leq i \leq n)$. La transition T entre l'état $s = ((q_1, v_1), \ldots, (q_n, v_n) : x_1, \ldots, x_p)$ et l'état $s\prime = ((q\prime_1, v\prime_1), \ldots, (q\prime_n, v\prime_n) : x\prime_1, \ldots, x\prime_p)$ peut être soit une émission(RS), une réception(RR), une exécution d'une action locale(RL), un passage de temps(RP), une synchronisation par rendez-vous(RS). La sémantique du système S est définie par la plus petite relation de transition satisfaisant les règles suivantes :*

1. (RE règle d'émission)
$$\frac{\big((q_i,v_i),a!v,\ \big(q'_i,v'_i\big)\big)\in T_D \quad x_j \text{ mot du canal } a}{(\ldots,(q_i,v_i),\ldots:\ldots,x_j,\ldots)\xrightarrow{a!v}(\ldots,\big(q'_i,v'_i\big),\ldots:\ldots,x_j.v,\ldots,)}$$

2. (RR règle de réception)
$$\frac{\big((q_i,v_i),\ a?x,\ \big(q_i',v_i'\big)\big) \in T_D \qquad x_j \text{ mot du canal } a}{(...,(q_i,v_i),...:...,x_j.v,...)\overset{a?x}{\rightarrow}\big(...,\big(q_i',v_i'\big),...:...,x_j,...\big)}$$

3. (RL exécution d'une action locale, c'est à dire n'affecte pas le reste du système)
$$\frac{\big((q_i,v_i),\ \alpha,\ G,\ D,\ z,\ \big(q_i',v_i'\big)\big) \in T_D \qquad v\models G}{(...,(q_i,v_i),...:...,x_i,...)\overset{\alpha}{\rightarrow}\big(...,\big(q_i',v_i'\big),...:...,x_i,...\big)}$$
où G est une contrainte temporal ou *garde* , D est l'échéance correspondante qui exige, au moment de sa satisfaction, que l'action α doit être tirée, z l'horloge qui doit être réinitialisée.

4. (RP passage de temps)
$$\frac{d\in R^+ \qquad \forall d'\leq d \qquad (v_i+d)\ \not\models D}{(...,(q_i,v_i),...:x_1,...,x_p)\overset{d}{\rightarrow}(...,(q_i,v_i+d),...:x_1,...,x_p)}$$

5. (RS pour cette règle les canaux x_i et x_j ne contiennent aucun messages)
$$\frac{\big((q_i,v_i),\ a?x,\ \big(q_i',v_i'\big)\big) \in T_D \quad \big((q_j,v_j),\ a!v,\ \big(q_j',v_j'\big)\big) \in T_D\ i\neq j}{(...,(q_i,v_i),...,(q_j,v_j),...:x_1,...,x_p)\overset{\tau}{\rightarrow}\big(...,\big(q_i',v_i'\big),...\big(q_j',v_j'\big),...:x_1,...,x_p\big)}$$

Remarque 4.1 *Cette sémantique prend en charge aussi bien les communications asynchrones (Règles RE et RR) que les communications synchrones (Règle RS).*

4.2.3 Exemple : Echange de message (communication)

Soit le processus E qui reçoit l'information(Rq_Data) sur le canal a, puis il attend pour une période de t unités de temps. Le processus offre le message m sur le canal b pour une période de d unités de temps, et termine son exécution en exécutant le processus $exit$.

L'expression de comportement pourrait être spécifiée comme suit

$$\begin{aligned} E ::= \quad & a?Rq\_Data \\ & (\Delta^t(b!m\{d\}; exit)) \end{aligned}$$

Dans l'état initial aucune action n'est en cours d'exécution, ce qui explique pourquoi l'ensemble des conditions des durées est vide. Nous correspondons dans cet état l'expression E pour former la configuration initiale du C-DATA ${}_{\phi}[E]$ où aucune action a été établie. De cette configuration, la première action à exécuter par le comportement E est la réception de l'information Rq_Data sur le canal a. L'horloge x, choisie par la fonction *get* à partir de l'ensemble $\mathcal{M}$ et ayant la valeur initiale 0, est associé à cette action. Nous appliquons la règle de réception(RR) du modèle C-DATA :

$$\underbrace{{}_{\phi}[E]}_{config0} \xrightarrow{\phi, a?Rq\_Data, x} \underbrace{{}_{\{x\geq d_1\}}[\Delta^t((b!m\{d\}\,;\, exit))]}_{config1}$$

d_1 : représente le temps de réception du message Rq_Data sur le canal a. Quand la réception est terminée, de la configuration $config1$ l'expression de comportement $\Delta^t((b!m\{d\}\,;\,exit))$ exprime que l'offert $b!m\{d\}$ ne peut pas être sensibiliser que si t unités de temps est écoulé.

$$config1 \quad \xrightarrow{t} \quad \underbrace{\phi[(b!m\{d\}\ ;\ exit)]}_{config2}$$

A partir de cette configuration, l'émission du message m sur le canal b, est possible

$$config2 \quad \xrightarrow{C_\phi \leq d, b!m, x} \quad \underbrace{\{x \geq d_2\}[\ exit]}_{config3}$$

d_2 : représente le temps d'émission du message m sur le canal b. De la sémantique de l'opérateur de préfixage ";", le processus $exit$ ne peut pas commencer son exécution que si le message m a été envoyé.

Nous pouvons avoir une configuration différente de $config2$, si le temps écoulé depuis la sensibilisation de l'action d'émission (calculer par l'horloge C_ϕ) dépasse l'échéance de l'offre d unités de temps, ainsi, le processus est transformé au processus $stop$.

$$\frac{C_\phi > d}{config2 \quad \xrightarrow{C_\phi} \quad \underbrace{\phi[\ stop]}_{config4}}$$

De la configuration $config3$, nous obtenons :

$$config3 \quad \xrightarrow{\{x \geq d_2\}, \delta, x} \quad \underbrace{\{x \geq 0\}[stop]}_{config5}$$

4.3 Mobile C-DATA

Avec le développement des systèmes réparties temps réel, de nouveaux besoins sont apparus parmi lesquels le besoin de maintenance, d'évolution et de reconfiguration des systèmes existants.

Dans la section précédente nous avons définit une extension des DATA's, qui prend en compte la communication à distance entre les différentes localités du calcul, pour exprimer la distribution du système. Cependant ce modèle demeure statique et ne permet pas une évolution dynamique du calcul. De ce fait, une extension de ce modèle s'impose C-DATA[MSK11], permettra la prise en compte de la création et la suppression des localités, ainsi que la migration des processus entre les localités.

Notre modèle distingue entre deux types de calculs, le calcul local et le calcul global. Le calcul local fait intervenir des communications entre processus de même localité. Ces communications se traduisent par des synchronisations de portées locales (interne) par rapport

aux autres localités. La description du système global peut être faite en utilisant le modèle C-DATA définit dans la section précédente.

Dans[MSK12], nous avons proposé une extension de C-DATA pour exprimer, la mobilité des processus, la création et la suppression des localités, la prise en charge de l'aspect dynamique du calcul.

4.3.1 Définitions préliminaires

Définition 4.8 *(actions) L'ensemble des actions dans le système est défini comme suit :*
- L'ensemble $Act = \mathcal{G} \cup \{i, \delta, go, create\}$
- L'ensemble des actions de communication entre les localités : l'émission ou la réception de messages via les canaux de communication $Act_{com} ::= a!m \mid a?x \mid \tau$ *(action d'émission, les actions de réception et de l'action silencieuse).*

Définition 4.9 *Un Mobile C-DATA est un septuple* $A(S, L_S, s_0, \vartheta, H, \Pi, T_D)$ *tel que :*

- S est un ensemble fini d'états,
- $L_S : S \rightarrow 2_{fn}^{\Phi_t(H)}$ est une fonction qui fait correspondre à chaque état s l'ensemble F des conditions de terminaison des actions potentiellement en exécution dans s,
- $s_0 \in S$ est l'état initial,
- ϑ l'esemble de l'alphabet des canaux de communication sur lesquels les messages transitent entre les sous-systèmes (DATA),
- H est un ensemble fini d'horloges,
- $\Pi = Act_{com} \cup Act$, est l'ensemble des actions locales et de communication,
- $T_D \subseteq S \times 2_{fn}^{\Phi_t(H)} \times 2_{fn}^{\Phi_t(H)} \times \Pi \times H \times S$ est l'ensemble des transitions.

Dans ce modèle une transition $(s, G, D, \alpha/(a(!/?)v)/\tau, z, s\prime)$ représente le passage de l'état s à l'état s', en lançant l'exécution de l'action $\alpha \in Act$ ou action de communication (émission ou réception) ou synchronisation dans une communication (action silencieuse) et en réinitialisant l'horloge z. G est la contrainte correspondante, qui doit être satisfaite pour tirer cette transition. D est l'échéance correspondante qui exige, au moment de son expiration, que l'action α soit tirée. $(s, G, D, \alpha/(a(!/?)v)/\tau, z, s\prime)$ peut être écrite $s \xrightarrow{G,D,\alpha/(a(!/?)v)/i),z} s'$.

Définition 4.10 ***(Système)*** *Un système* $S = (A_1, \ldots, A_n)$*, est un Mobile C-DATA, tel que chaque* $A_i = (S_i, L_{S_i}, s_{0_i}, \vartheta, H_i, \Pi_i, T_{iD})$ *est un DATA.*

On remarque qu'un *Mobile C-DATA*, est structurellement un *C-DATA*, sur lequel les opérations de création et suppression de localités et migration de processus sont définies.

4.3.2 Construction opérationnelle des Mobile C-DATA's :

Définition 4.11 *Soit* $S = (A_1, \dots A_n)$ *un Mobile C-DATA, telque chaque* $A_i = (S_i, L_{S_i}, s_{0_i}, \vartheta, H_i, \Pi_i, T_{iD})$*. La sémantique du système* S *est définie par les règles suivantes. Une transition* T *entre l'état* $s = ((q_1, v_1), \dots, (q_n, v_n) : x_1, \dots, x_p)$ *et l'état* $s\prime = ((q\prime_1, v\prime_1), \dots, (q\prime_n, v\prime_n) : x\prime_1, \dots, x\prime_p)$ *est une règle d'émission* (RE) *ou réception* (RR) *ou d'exécution d'une action locale*(RL) *ou passage de temps*(RP) *ou règle de création*(RC) *ou règle de migration*(RM) *ou règle de suppression*(RD) *ou règle de synchronisation*(RS) *:*

(1) (RE règle d'émission)
$$\frac{\big((q_i,v_i),a!v,\ \big(q'_i,v'_i\big)\big)\in T_D}{(\dots,(q_i,v_i),\dots:\dots,x_j,\dots)\overset{a!v}{\to}\big(\dots,\big(q'_i,v'_i\big),\dots:\dots,x_j.a,\dots,\big)}$$

(2) (RR règle de réception)
$$\frac{\big((q_i,v_i),\ a?x,\ \big(q'_i,v'_i\big)\big)\in T_D}{(\dots,(q_i,v_i),\dots:\dots,x_j.a,\dots)\overset{a?x}{\to}\big(\dots,\big(q'_i,v'_i\big),\dots:\dots,x_j,\dots\big)}$$

(3) (RL règle d'exécution d'une action locale)
$$\frac{\big((q_i,v_i),\ a,\ G,\ D,\ z,\ \big(q'_i,v'_i\big)\big)\in T_D \qquad v\models G}{(\dots,(q_i,v_i),\dots:\dots,x_i,\dots)\overset{\alpha}{\to}\big(\dots,\big(q'_i,v'_i\big),\dots:\dots,x_i,\dots\big)}$$
où G est une contrainte temporale ou *garde*, D est l'échéance correspondante qui exige, au moment de sa satisfaction, que l'action α soit tirée, z l'horloge qui doit être réinitialisée.

(4) (RP passage de temps)
$$\frac{d\in R^+ \qquad \forall d'\leq d \qquad (v_i+d)\ \not\models D}{(\dots,(q_i,v_i),\dots:x_1,\dots,x_p)\overset{d}{\to}(\dots,(q_i,v_i+d),\dots:x_1,\dots,x_p)}$$

(5) (RC règle de création)
$$\frac{(Q(\dots,(q_n,v_n)),\ create(l,E),\ Q'(\dots,(q_{n+1},v_{n+1})))\in T_D}{(\dots,(q_n,v_n):x_1,\dots,x_p)\overset{create}{\to}(\dots,(q_{n+1},v_{n+1}):x_1,\dots,x_{p+1})}$$

(6) (RM règle de migration)
$$\frac{\big((q_i,v_i),\ go(l,E),\ (q'_i,v'_i)\big)\in T_D\ \ v_i\models G}{(\dots,(q_i,v_i),\dots:x_1,\dots,x_p)\overset{go}{\to}(\dots,\big(q'_i,v'_i\big),..:x_1,\dots,x_p)}$$

(7) (RD règle de suppression)
$$\frac{(Q(\dots,(q_n,v_n)),\ \delta,\ Q'(\dots,(q_{n-1},v_{n-1})))\in T_D}{(\dots,(q_n,v_n):x_1,\dots,x_p)\overset{\delta}{\to}(\dots,(q_{n-1},v_{n-1}):x_1,\dots,x_{p-1})}$$

(8) (RS pour cette règle, les canaux x_i and x_j ne contiennent aucun messages)
$$\frac{\big((q_i,v_i),\ a?x,\ (q'_i,v'_i)\big) \in T_D \quad \big((q_j,v_j),\ a!v,\ (q'_j,v'_j)\big) \in T_D\ i{\neq}j}{(...,(q_i,v_i),...,(q_j,v_j),...{:}x_1,...,x_p) \xrightarrow{\tau} \big(...,(q'_i,v'_i),...,(q'_j,v'_j),...{:}x_1,...,x_p\big)}$$

4.4 Transformation formelle des DATA* en automates temporisés

L'objectif de la modélisation des systèmes réactifs en général, et en particulier les systèmes temps réel par des modèles et langages basés sur une sémantique de vrai parallélisme (sémantique de maximalité) est la vérification quantitative des propriétés telles que le respect des contraintes temporelles et des performances minimales. Dans ce contexte, le modèle des DATA* étant utilisé pour exprimer la sémantique du langage D-LOTOS. Cependant il n'existe pas d'outils de vérification basés sur ce modèle. Afin de tirer profit des outils existant et qui opère sur les automates temporisés nous proposons un algorithme permettant le passage des DATA* en automates temporisés[MS11]. Notre approche consiste à interpréter les comportements décrits par des DATA*'s par des automates temporisés. La principale différence entre les deux modèles est l'hypothèse sur les actions, dans les automates temporisés les actions sont supposées atomiques et ont des durées nulles, pour les DATA* les actions sont supposées non atomique et de durées non-nulle, par l'utilisation de la sémantique de vraie parallélisme.

Dans la partie outillage, nous avons développé un outil de compilation des spécifications en D-LOTOS vers les DATA*. L'outil $DLOTOS2DATA$[MSB12] prend en entrée une spécification D-LOTOS, cette spécification est vérifiée sur le plan lexicale, syntaxique et sémantique, par la suite le DATA* correspondant est généré en appliquant les règles de définition des DATA*s.

4.4.1 Approche de transformation

Dans[SBG11], il a été introduit une méthodologie d'interprétation des DATA*'s dans les automates temporisés. Cette interprétation permet de vérifier les comportement exprimés par des DATA* à l'aide du modèle checker UPPAAL[LDBY03].

Prenons l'expression de comportement $E = a; b; stop$, en supposant que les actions a et b ont des durées non nulles respectives 10 et 12. Supposons que l'action a ne peut commencer que dans les trois premières unités de temps et b dans les quatre unités de temps, le comportement de E est donné par la figure 4.2

A partir de l'état initial s_0, l'action a peut commencer son exécution, tout en respectant la condition ($c_\phi \leq 3$), ce qui signifie que a peut toujours commencer son exécution si une horloge particulière (c_ϕ) n'a pas atteint la valeur 3 unités de temps depuis la sensibilisation du système. Le système passe à l'état s_1 et ne peut pas quitter cet état avant la fin de l'action a, dans cet état l'action a est potentiellement en exécution. Le même raisonnement s'applique à l'action b.

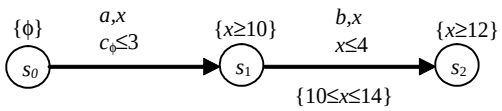

FIG. 4.2 – DATA* pour l'exécution séquentielle

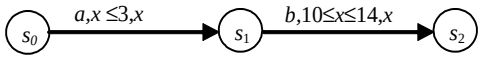

FIG. 4.3 – Automate temporisé correpondant au DATA*

La transition entre l'état s_0 et l'état s_1 peut être représentée dans un automate temporisé par la transition $s_0 \overset{begin(a),c_\phi \leq 3,x}{\longrightarrow} s_1$ (cette transition exprime le début de l'action a, la garde sur le délai de l'offre, et x est l'horloge à réinitialiser à zéro par cette transition pour comptabiliser le temps d'exécution de a). Une fois le système est à l'état s_1, il ne pourra pas le quitter ni avant la terminaison de a ni après l'expiration du délai d'offre de l'action b, l'action b est sensibilisée au moment de la terminaison de a, c'est à dire le délai d'offre de b commence son expiration une fois a est terminée. Là également, l'expression du passage de s_1 à s_2 pourra se faire à l'aide de la transition $s_1 \overset{begin(b),10\leq x\leq 14,x}{\longrightarrow} s_2$.

Il est clair qu'il est possible d'exprimer les durées sans être obliger de considérer chaque action comme deux évènements : début et fin.

L'approche consiste à modéliser une action qui a deux paramètres optionnels : un délai de l'offre et une durée d'exécution, avec un automate temporisé contenant :

- Une transition, pour exprimer le début de l'action (avec une garde s'il existe, un délai d'offre à respecter et une réinitialisation de l'horloge appropriée pour capturer l'instant de début de l'action),
- Un état, où le système réside au cours de l'exécution de l'action (le système n'étant pas forcé de quitter cet état une fois que l'action se termine).

La garde de la prochaine transition est utilisée pour capturer à la fois, la fin de l'action et le début de la prochaine action. Finalement nous obtenons l'automate représentant le comportement global de E dans la figure 4.3.

En comparant les deux modèles illustrés par les deux figures 4.2 et 4.3, on constate que les deux expriment le même comportement sauf que pour l'automate temporisé on perd l'information sur la fin de l'action b.

Il est évident qu'il est impossible de capturer la fin de la toute dernière action sans ajouter une nouvelle transition. Il a été proposé dans l'approche [SBG11], que tous les processus

exprimés par les DATA*'s utilisent obligatoirement l'action particulière atomique δ pour marquer leurs fins. Et ainsi on obtient le comportement illustré par la figure 4.4.

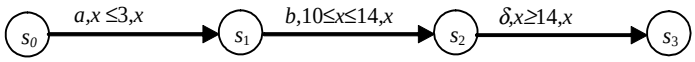

FIG. 4.4 – Automate temporisé avec terminaison

Le même raisonnement s'applique pour l'interprétation du parallélisme et l'urgence des actions par les automates temporisés. Pour plus de détaille voir[SBG11].

4.4.2 Algorithme de transformation des DATA*'s en automates temporisés

Dans cette section, nous proposons un algorithme de transformation des DATA*'s en automates temporisés [MSB12]. L'objectif de cette transformation est l'exploitation des automates temporisés pour la vérification formelle en utilisant le modèle checker UPPAAL.

$DATA * 2TA$ Transformation des **DATA*** vers **TA**

Input : $\mathcal{DATA}*\ (S, L_s, s_0, \mathcal{H}, \mathcal{T}_D)$

Output : $\mathcal{TA}(Q, Act, q_0, X, E, Inv)$

$q_0 := s_0$; // état initial
$Q := \{q_0\}$; // ensemble des états
$X := \phi$ // ensemble des horloges
$Act := \phi$ //alphabet des actions

For all *transition* $t \in \mathcal{T}_\mathcal{D}$

// t : transition de la forme $s_i \overset{G,D,\alpha,x}{\longrightarrow} s_{i+1}$

//initialisation des paramètres de chaque transition
// et les ensembles de l'automates temporisés
$q_s := t.s_i$;
$q_t := t.s_{i+1}$;
$a := t.\alpha$;
$x := t.x$;
$Q := Q \cup \{q_s, q_t\}$;
$Act := Act \cup \{a\}$
$X := X \cup \{x\}$;
$G := t.G; D := t.D$;

if $(D = \emptyset)$ **then**

//l'action courante n'est pas urgente
$I := \emptyset$;
// Renommer les horloges dans la garde G par x
$renameClocks(G, x)$
$Condition := G$;
else
$I := Invariant(G, D)$;
$condition := G/D \wedge P_i(D)$
endif
$e := (q_s, q_t, a, x, condition)$;
$Inv := Inv \cup \{(I, q_t)\}$;
$E := E \cup e$;
endfor
// La denière transition : action δ
//Récupérer la durée d'exécution de la dernière action dans le modèle
$C := ExtractDuration(q_t, L_s)$;
$Act := Act \cup \{\delta\}$
$x := get(\mathcal{H} \backslash X)$ //horloge pour δ
$X := X \cup \{x\}$;
$E := E \cup (q_t, q_\delta, \delta, x, C)$;
//En sortie l'automate temporisé $\mathcal{TA}$

$\mathcal{TA} := (Act, Q, q_0, X, E, Inv)$
return $\mathcal{TA}$;
end.

Fonctionnement de l'algorithme

L'algorithme de transformation des DATA*'s en automate temporisé prend en entrée un DATA* représentés par : un ensemble d'états S, une fonction L_s qui associe à chaque état s l'ensemble F des conditions de terminaison des actions potentiellement en exécution dans s, un état initial s_0, un ensemble fini d'horloges, et $\mathcal{T}_d$ ensemble des transitions de la forme $s \overset{G,D,\alpha,x}{\longrightarrow} s'$.

La contrainte temporelle G (respectivement D) est décrite comme suit :

$\wedge d_i \sim x \sim d_s$ tels que : d_i, $d_s \in \mathbb{R}$ et $\sim \in \{<, \leq\}$ et $d_i \sim x$, $x \sim d_s \in G$ (respectivement D).

Pour la contrainte d'urgence D ($D \neq \phi$) :

$G/D = \wedge d_i \sim x \sim d_s$ tel que $\wedge d_i \sim x \sim d_s \models G \wedge \overline{D}$

$P_i(D) = \wedge d_i \sim x$

$P_s(D) = \wedge x \sim d_s$

En sortie, l'algorithme produit un automate temporisé construit comme suit :

- Au début, l'état initial de l'automate temporisé, il est le même état initial du DATA*
- Pour chaque transition de la forme $s \overset{G,D,\alpha,x}{\longrightarrow} s'$ dans le DATA* : L'algorithme commence par l'initialisation de l'état de départ s, l'état d'arrivé s', l'action courante α et l'horloge x.
- Récupérer les deux contraintes temporelles : G (garde), D (deadline)
- Mise à jour des ensembles (ensemble des états Q, Act des actions, H : ensemble d'horloges), et l'ensemble E des transitions de $\mathcal{TA}$ $< s, s', a, x, G/D \wedge P_i(D) > \in E$, l'invariant $I(s) = P_s(D)$

Si l'action en cours n'est pas urgente alors la contrainte d'urgence D est vide. La construction de la transition de l'automate temporisé est faite comme suit :

- Les deux états de départ et d'arrivée sont déjà initialisés
- Absence de l'invariant sur l'état d'arrivée
- La garde G de $\mathcal{DATA}*$ devient une condition sur la transition

La fonction $renameClocks(G, x)$, permet de renommer l'horloge de type c_ϕ dans le $\mathcal{DATA}*$ par l'horloge x dans la contrainte G. Du fait que le temps d'une horloge dans un $\mathcal{DATA}*$ ne peut pas s'écouler avant que l'attribution de cette horloge à une action soit faite, le système est augmenté avec l'horloge c_ϕ (initialisée dés la sensibilisation du système) pour comptabiliser le temps écoulé avant le début de toute action.

D'autre part, les horloges d'un automate temporisé commencent à comptabiliser le temps dès la sensibilisation du système. Donc, on peut omettre l'horloge c_ϕ et a remplacer par x sans affecter la sémantique du modèle.

Si l'action est urgente : L'invariant est calculé à partir de la contrainte d'urgence D et de la garde G.

Dans la dernière étape, il est toujours nécessaire d'ajouter l'action δ à l'automate temporisé pour capturer la fin de la dernière action du $\mathcal{DATA}*$. La fonction $ExtractDurtion(q_t, L_s)$ sert à calculer la durée de la dernière action.

Enfin, l'algorithme retourne l'automate temporisé composé de :

- L'ensemble Act : toutes les actions du DATA* et l'action particulière δ.
- L'ensemble Q : les états qui composent le DATA* plus l'état q_δ
- q_0 : l'état initial.
- X : l'ensemble des horloges.
- E : l'ensemble des transitions de l'automate temporisé
- Inv : ensemble de couple de la forme (q, c) tels que q représente l'état de l'automate temporisé et c la condition sur cet état.

Terminaison

Le théorème suivant établit la terminaison de l'algorithme $DATA*2TA$, la preuve de terminaison de l'algorithme est une étape essentielle pour sa validité, pour la production de l'automate temporisé.

Théorème 4.1 *(Terminaison) l'algorithme $DATA*2TA$ termine*

Preuve. Pour démontrer ce point, rappelons que le DATA* en entrée représente un modèle sémantiques fini, c'est à dire le nombre d'état dans le modèle est déterminé et fini. Remarquons que dans l'algorithme chaque état du DATA* est traité une et une seule fois, de ce fait tous les états du DATA* finiront par être traités. D'où le résultat. ■

Complexité

Nous montrons la complexité en temps de notre algorithme de transformation $DATA*2TA$

Théorème 4.2 (***Complexité de DATA*2TA***) *Etant donné un DATA*, la transformer du DATA* en automate temporisé est un problème linéaire en nombre de transitions.*

Preuve. Nous considérons un DATA* (qui ne contient ni contraintes diagonales ou des contraintes de mise à jour). La taille de l'automate temporisé construit est la même que la taille du DATA* plus la transition de l'action δ et le dernier état. Les transitions de DATA* une à une, ce qui donne un temps de génération du l'automate temporisé linéaire par rapport au nombre de transitions du DATA*. ■

4.5 Conclusion

Dans ce chapitre, nous avons introduit et défini formellement deux modèles sémantiques basés sur la maximalité appelés respectivement Communicating DATA(C-DATA) et mobile C-DATA. Le premier modèle est destiné principalement à la spécification de systèmes, temps-réel, communicants, c'est à dire les systèmes qui peuvent être vus comme un ensemble d'automates qui communiquent via des canaux de communications. Ce modèle s'inspire de deux modèles à savoir le modèle des automates communicants et le modèle des DATA's. Le deuxième modèle étant une extension des C-DATA par le concept de mobilité. Le modèle est ainsi appelé *Mobile C-DATA*.

Par la suite, nous avons proposé un algorithme de passage du modèle des DATA* vers les automates temporisés. Nous avons montré aussi la terminaison de l'algorithme et sa complexité.

Il est clair que le modèle des C-DATA constitue un domaine d'interprétation de tout langage décrivant conjointement les concepts de contraintes temporelles, durées d'actions et distribution. Dans le chapitre 5, nous définissons un langage formel de cette catégorie. De même dans le chapitre 6, le modèle Mobile C-DATA sera utilisé comme domaine d'interprétation d'un langage plus étendus et qui considère l'aspect de mobilité.

Chapitre 5

Distributed D-LOTOS : Un modèle de spécification des systèmes distribués.

Sommaire

Ce chapitre, qui s'appuie en grande partie sur les résultats de [MSK11] et de [MS09], présente un nouveau calcul de processus distribué temps réel (DD-LOTOS). Dans un système réparti, les localités peuvent représenter les agents, ou des machines sur lesquelles les agents migrent et s'exécutent. D'autre part dans les algèbres de processus temps réel (RT-LOTOS, ET-LOTOS, D-LOTOS, etc) étudiées dans le chapitre 3, nous constatons qu'il n'y a pas de notion de localité, d'où l'incapacité pour la spécification de la répartition du calcul[GNS00]. Notre calcul de processus vise à pallier les limitations du langage D-LOTOS, en fournissant un modèle de programmation réparti, temps réel qui autorise la programmation explicite des localités.

Nous présentons d'abord la syntaxe, suivie par la sémantique opérationnelle du modèle. Ensuite, plusieurs exemples seront développés sur le modèle.

Du D-LOTOS, nous retenons les aspects liés au temps tel que durées des actions, et opérateurs expriment les contraintes temporelles.

5.1 Présentation de DD-LOTOS

La motivation majeure de notre travail est de fournir un modèle abstrait pour la programmation distribuée. Nous nous intéressons aux modèles de processus qui représentent explicitement l'existence d'objets, nommés localités, domaine ou site, qui abstraient les lieux physiques, comme les ordinateurs, ou logiques, comme les agents. Une localité est un environnement d'activité de calcul ; les processus exécutent des actions et sont les responsables de l'évolution globale du système. Intuitivement, une localité héberge les ressources locales que les agents mobiles utilisent pour interagir les uns avec les autres. Pour leur permettre de remplir tous ces rôles, les localités sont nommées et structurées selon une structure plate, c'est une manière de programmer les localités explicitement. Ainsi une localité typique peut être décrite dans la figure 5.1 :

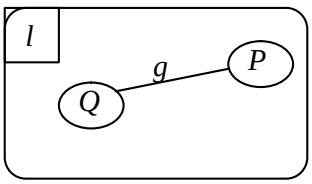

FIG. 5.1 – Localité

P, Q représentent deux processus qui résident dans la localité l, et peuvent communiquer via la porte g. Un système distribué peut être vu comme une collection de localités indépendantes.

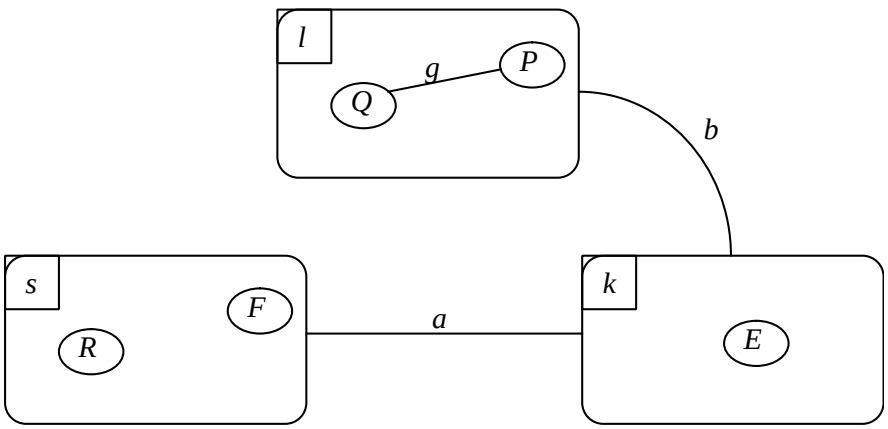

FIG. 5.2 – Système distribué

La figure 5.2, présente un système distribué composé de trois localités l, k et s. La localité l communique avec la localité k via le canal b, et la localité k communique avec la localité s via le canal a. La communication entre les localités est autorisée par le principe d'échange de messages, c'est à dire deux activités sur deux localités différentes communiquent seulement

par échange de messages.

L'expression de comportement $l(E)$ dénote l'exécution du comportement E dans la localité l. Donc, un système peut être considéré composé de plusieurs activités exécutées dans plusieurs localités. Par exemple le système de la figure 5.2, est représenté par l'expression de comportement suivante :

$$l(Q|[g]|P) \mid k(E) \mid s(R|F)$$

Chaque activité évolue localement, pour cette raison, la portée des actions est limitée à la localité correspondante. Comme pour LOTOS, les noms des actions dénotent également les portes de communication. Les portes ne sont pas accessibles par des activités à distance, le seul moyen de communication à distance est les canaux de communication définis par l'utilisateur du système.

Nous notons que la synchronisation entre les activités dans la même localité est autorisée. Par exemple l'expression de comportement suivante :

$$l(g; P|[g] \mid \ g; \ Q)$$

Spécifie les activités $g; P$ et $g; Q$ qui sont dans la localité l, et qui peuvent se synchroniser sur la porte g, comme suit :

$$l \ (g, \ P \mid [g] \mid g, \ Q) \xrightarrow{g} l \ (P \mid [g] \mid Q).$$

De l'autre côté deux activités sur deux localités différentes peuvent communiquer en échangeant des messages sur les canaux de communication. Cette communication à distance est asynchrone non-bloquante. Nous avons étendu notre modèle par deux nouvelles primitives pour prendre en compte le paradigme d'échange de messages sur des canaux de communication : $a!m\{d\}$ et $a?x; E$. La première expression de comportement permet l'envoi du message m sur le canal a, cette action d'envoi est offerte à l'environnement pour une durée d unités de temps, la seconde expression de comportement exprime la réception d'un message sur le canal a.

5.1.1 Exemple introductif

Cet exemple nous permet d'introduire la notion de base de l'extension du temps : la durée d'une offre d'une action d'envoi. En effet, si nous analysons le comportement d'un émetteur, on constate que, après avoir transmis un message, il est prêt à attendre un accusé de réception pendant un certain temps (notion de durée d'une offre) ou de retransmettre le message lorsque ce temps s'écoule (notion de retard). Cette description peut être traduite par l'expression de comportement suivante :

$$transmit!m\{d\}; (ack\{wt\}; P[]\Delta^{wt}Q)$$

L'expression de comportement $transmit!m\{d\}$ tel que $d \in D$, signifie que l'action d'envoi $transmit!m$ ne sera pas offerte après d unités de temps, en d'autres termes cette action ne peut se produire que dans l'intervalle de temps $[0, d]$.

5.1.2 Syntaxe

Pour prendre en considération les localités et la communication à distance, la syntaxe D-LOTOS est étendue comme suit :

$E ::=$	...	**Comportements**
	$\mid a!v\{d\}; E$	**Envoyer un message**
	$\mid a?x; E$	**Recevoir un message**
$S ::=$		**Systèmes**
	ϕ	**Vide**
	$\mid S \mid S$	**Composition**
	$\mid l(E)$	**Activité *E* dans la localité** l

FIG. 5.3 – *Syntaxe du langage DD-LOTOS*

Définition 5.1 *(actions) : Les actions dans le système global sont :*

- L'ensemble des actions de communication entre les localités : L'envoi et la réception de messages via les canaux de communication $Act_{com} ::= a!m \mid a?x \mid \tau$ (action d'envoi ou de production, action de réception et l'action silencieuse).
- L'ensemble $Act = \mathcal{G} \cup \{i, \delta\}$.

Définition 5.2 *(Localités et canaux de communication) : L'ensemble $\mathcal{L}$ parcouru par l, désigne l'ensemble des localités. ϑ un ensemble infini de canaux définis par les utilisateurs parcouru par a, b, ... Les canaux sont utilisés pour l'échange de messages entre les localités.*

La syntaxe du calcul est donnée par la figure 5.3. Deux catégories d'expressions, la syntaxe des systèmes S et l'expression de comportement E. La sémantique informelle des expressions syntaxiques est la suivante :

- L'expression de comportement $a!v\{d\}; E$, spécifie l'envoi du message v via le canal de communication a. Cette opération d'envoi doit se produire dans l'intervalle temporel [0, d]. De l'autre côté, l'expression de comportement $a?x; E$, spécifie la réception d'un message sur le canal a, qui sera substitué à la variable x. Cette variable est utilisée dans l'expression de comportements E.

- Un système est défini par :
  - Vide, exprimé par ϕ,
  - La composition de sous-systèmes $S \mid S$, ou
  - Une expression de comportement E dans une localité l exprimée par $l(E)$.

5.1.3 Sémantique opérationnelle structurée :

La sémantique opérationnelle des comportements est donnée par la sémantique opérationnelle de D-LOTOS. Cette sémantique est étendue à DD-LOTOS en donnant les règles sémantiques pour les systèmes communicants comme suit :

Processus $a!v\{d\};E$: Prenons la configuration ${}_M[a!v\{d\};E]$, l'émission du message v commence une fois que toutes les actions indexées par l'ensemble M ont terminé leur exécution, ce qui est conditionné par le prédicats $Wait(M)$ qui doit être égal à $false$ dans la règle 1. Les règles 2 et 3 expriment le fait que la prise en compte de l'écoulement du temps dans $a!v\{d\};E$ commence uniquement si toutes les actions référencées par M terminent leur exécution. La règle 4 impose que l'occurrence de l'action d'envoi ait lieu dans la période d, dans le cas contraire le processus est transformé en *Stop*.

1. $\dfrac{\neg Wait(M)}{{}_M[a!v\{d\};E] \xrightarrow{M^{a!v}x} {}_{\{x:a!v:t\}}[E]} \qquad x = get(\mathcal{M})$

2. $\dfrac{Wait(M^{d'}) \; or \; (\neg Wait(M^{d'}) \; and \; \forall \varepsilon{>}0.\; Wait(M^{d'-\varepsilon})) \qquad d'{>}0}{{}_M[a!v\{d\};E] \xrightarrow{d'} {}_{M^{d'}}[a!v\{d\};E]}$

3. $\dfrac{\neg Wait(M)}{{}_M[a!v\{d'+d\};E] \xrightarrow{d} {}_M[a!v\{d'\};E]}$

4. $\dfrac{\neg Wait(M) \; and \; d'{>}d}{{}_M[a!v\{d\};E] \xrightarrow{d'} {}_M[stop]}$

Processus $a?x;E$: Prenons la configuration ${}_M[a?x;E]$, la règle suivante exprime que la réception commence une fois que les actions indexées par l'ensemble M terminent leur exécution.

$$\frac{\neg Wait(M)}{{}_M[a?x;E] \xrightarrow{M^{a?x}y} {}_{\{y:a?x:0\}}[E]}$$

Communication à distance

Les activités distribuées échangent les messages entre eux, l'expression de comportement $l(a!v\{d\})$, exprime que le message v est offert pour une durée d unités de temps, cette activité évolue dans la localité l, ce message doit être envoyé via le canal a. De l'autre côté, $k(a?xE)$ spécifie que l'activité E évolue dans la localité k, est prête à recevoir un message via le canal a. La règle suivante définit la communication à distance entre les deux activités distribuées

via le canal a. Dans ce cas, la communication est traduite par une évolution interne (action silencieuse τ) :

$$\frac{-}{{}_{M}[l(a!v\{d\};E1)] \mid {}_{M'}[k(a?xE2)] \xrightarrow{\tau} {}_{M}[l(E1)] \mid {}_{M'}[k(E2\{v/x\})]}$$

L'évolution temporelle du système

$$\frac{E \xrightarrow{d} E'}{l(E) \xrightarrow{d} l(E')}$$

$$\frac{S_1 \xrightarrow{d} S_1' \qquad S_2 \xrightarrow{d} S_2'}{S_1 \mid S_2 \xrightarrow{d} S_1' \mid S_2'}$$

Le temps agit sur tout le système.

5.2 Exemple

Prenons l'exemple d'une version simplifiée d'un protocole de communication dans un réseau local représenté par la figure 5.4. Le système est composé d'un émetteur de données et deux de récepteurs.

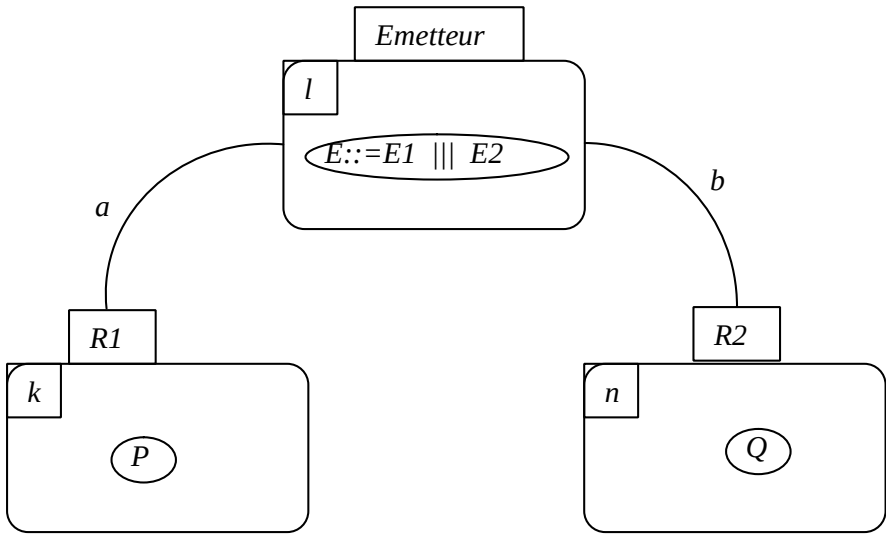

FIG. 5.4 – Système d'un émetteur et deux récepteurs

L'émetteur envoie des messages sur le réseau pour les deux récepteurs, d'autre part, en cas de perte de message, le récepteur concerné renvoie un acquittement négatif à l'émetteur. Si l'émetteur reçoit un acquittement négatif, il renvoie le message perdu. L'émetteur envoie périodiquement les messages de données. Ce système peut être défini par $l\ (E) \mid k\ (P) \mid n(Q)$ où :

- l est la localité de l'émetteur, et E son expression de comportement,
- k est la localité du récepteur $R1$, et P son expression de comportement,

- n est la localité du récepteur $R2$, et Q son expression de comportement.

Nous supposons que l'émetteur communique avec le récepteur $R1$ via le canal a, et avec le récepteur $R2$ via le canal b.

Le comportement de l'émetteur est de la forme :

1. Les messages sont envoyés chaque t unités de temps. Chaque message est envoyé aux récepteurs via les canaux a et b.
2. Lorsque l'émetteur reçoit un accusé de réception négatif à partir d'un récepteur, il procède à la retransmission du message perdu au récepteur concerné.

Le comportement de chaque récepteur est de la forme :

1. A Chaque réception d'un message v, il est concaténé avec les messages déjà reçus,
2. Lorsqu'il y aura une détection d'une perte d'un message, le récepteur envoi un acquittement négatif à l'émetteur.

Dans la localité l, l'activité E est composée de deux sous-activités, qui sont spécifiées respectivement par les expressions de comportement $(a!v\{d\} \mid b!v\{d\})$ et $c?xE'$. L'expression de comportement $(a!v\{d\} \mid b!v\{d\})$ décrit le fait que l'émetteur peut envoyer le message v sur les deux canaux a et b, sous la contrainte, que cette offre de message est pour une durée de d unités de temps.

Dans l'expression $c?xE'$ (c peut être a ou b), identifie le message perdu. c est le canal sur lequel on reçoit l'acquittement négatif. $E\prime$ spécifie l'opération de renvoie. Ainsi l'expression de comportement E peut être définie par :

$$E ::= E1 \ ||| \ E2$$

$$E1 ::= (a!v\{d\}|b!v\{d\}) >> \ \Delta^t E1$$

$$E2 ::= (c?xE') >> E2$$

P et Q sont respectivement les comportements associés aux localités k et n ($R1$ et $R2$).

$$P ::= (a?xB \ [] \ \Delta^t i; c!Nack\{d\}) >> P$$

$$Q ::= (b?xB \ [] \ \Delta^t i; c!Nack\{d\}) >> Q$$

B spécifie l'opération de concaténation des messages reçus. Le paramètre d est lié aux caractéristiques du réseau.

Specification $Sender2receivers[a, b]$
$l\ (E) \mid k\ (P) \mid n(Q)$.

Where
process $E ::= E1\ \ |||\ \ E2$
Where
$E1 ::= (a!v\{d\}|b!v\{d\}) >> \Delta^t E1$
$E2 ::= (c?xE') >> E2$
Endproc
process $P ::= (a?xB\ []\ \Delta^t i; c!Nack\{d\}) >> P$
Endproc
process $Q ::= (b?xB\ []\ \Delta^t i; c!Nack\{d\}) >> Q$
Endproc
Endspec.

FIG. 5.5 – *Spécification d'un émetteur et de deux récepteurs P et Q*

5.2.1 Spécification

La spécification complète dans le langage DD-LOTOS du système est donnée par la figure 5.5.

5.2.2 Génération du modèle C-DATA

Dans cette section, nous décrivons comment générer un C-DATA à partir d'une spécification DD-LOTOS. Prenons l'exemple précédent du protocole de communication simplifié. La spécification complète dans le langage DD-LOTOS est décrite par la figure 5.5.

Le système se compose de trois sous-systèmes, qui résident dans trois localités différentes. Dans une localité, l'activité représente un DATA*. La communication entre les activités des différentes localités est assurée par émission/réception des messages, le système global est représenté par le modèle C-DATA.

Dans la localité l, l'expression de comportement de la source, représente une composition parallèle de deux expressions, elle se traduit par :

$$_{\phi}[E] \rightarrow_{\phi} [E1] \ ||| \ _{\phi}[E2]$$

Dans l'état initial aucune action n'est en cours d'exécution, ce qui explique pourquoi l'ensemble des conditions des durées est vide. cet état est défini par la configuration initiale $_{\phi}[E]$. De cette configuration, l'émission du message v sur les canaux a et b is possible. L'horloge x, choisie par la fonction *get* à partir de l'ensemble $\mathcal{M}$ et ayant la valeur initiale 0, est associée à cette action d'émission. Nous appliquons la règle d'émission (RE) du modèle C-DATA

$$\underbrace{{}_{\phi}[E1] \;|||\; {}_{\phi}[E2]}_{config0} \xrightarrow{\phi, a!v, x_{\{x \geq t_1\}}} \underbrace{[b!v\{d\}) >> \;\Delta^t E1] \;|||\; {}_{\phi}[E2]}_{config1}$$

$$config1 \xrightarrow{b!v, y_{\{x \geq t_1, y \geq t_2\}}} \underbrace{[\;\Delta^t E1] \;|||\; {}_{\phi}[E2]}_{config2}$$

t_1(resp t_2) représente le temps d'émission du message v sur le canal a(resp b). Le même raisonnement est appliqué de la façon suivante à l'autre branche, où l'émission sur le canal b commence avant l'émission sur le canal a :

$$\underbrace{{}_{\phi}[E1] \;|||\; {}_{\phi}[E2]}_{config0} \xrightarrow{\phi, b!v, y_{\{y \geq t_2\}}} \underbrace{[a!v\{d\} >> \;\Delta^t E1] \;|||\; {}_{\phi}[E2]}_{config1}$$

$$config1 \xrightarrow{a!v, x_{\{x \geq t_1, y \geq t_2\}}} \underbrace{[\;\Delta^t E1] \;|||\; {}_{\phi}[E2]}_{config2}$$

Dans la configuration $config2$ l'expression de comportement [$\Delta^t E1$], vu la sémantique de l'opérateur de séquencement ">>", ne peut évoluer que si les deux émissions sur les canaux a et b terminent, en d'autres termes, si les conditions sur les durées sont satisfaits, ce qui correspond à la condition $\{x \geq t_1 \wedge y \geq t_2\}$, dans ce cas on reste dans la configuration $config2$. La transition suivante devient possible :

$$\underbrace{{}_{\{x \geq \tau_1, y \geq \tau_2\}}[\;\Delta^t E1] \;|||\; {}_{\phi}[E2]}_{config2} \xrightarrow{\{x \geq t_1, y \geq t_2\}} \underbrace{{}_{\phi}[\Delta^t E1] \;|||\; {}_{\phi}[E2]}_{config2}$$

Dans la localité k (resp n), l'expression de comportement du récepteur $R1$(resp $R2$), est un choix entre, soit recevoir un message sur le canal a(resp b), ou si la réception n'est pas effectuée après t unités de temps un message d'acquittement négatif est envoyé à la source et cela ce traduit dans le C-DATA par :

$$\underbrace{{}_{\phi}[P]}_{config0} \xrightarrow{\phi, a?v, x} \underbrace{[B[x/v] >> \;{}_{\phi}[P]}_{config1}$$

D'autre part, lorsque la durée de réception (t unités de temps) est dépassée, le numéro du message perdu ($Nack$) est transmis :

$$\frac{}{{}_{\phi}[P] \xrightarrow{d} \underbrace{[i; c!Nack\{d\} >> \;{}_{\phi}[P]}_{config2}} \; d \geq t$$

$$\frac{}{{}_{\phi}[P] \xrightarrow{d} \underbrace{[i; c!Nack\{d\} >> \;{}_{\phi}[P]}_{config2}} \; d \geq t$$

Le même raisonnement est appliqué dans la localité n.

5.3 Conclusion

Nous avons proposé une première extension du langage formel D-LOTOS avec primitives de communication à distance, et l'introduction du concept de localités dans le but d'exprimer la distribution du calcul. Notre langage nommé DD-LOTOS (*Distributed Durational LOTOS*) pour LOTOS distribué avec durées d'actions, est capable de spécifier les systèmes distribués et temps réel.

Dans ce qui suit, nous envisageons d'étendre notre langage pour supporter la mobilité, exprimer par la migration des entités entre les diverses localités du système, et la création, suppression des localités.

Chapitre 6

Mobile DD-LOTOS

Sommaire

Ce chapitre, qui s'appuie sur les résultats de[MSK12], présente un calcul de processus distribué, temps réel et mobile (mobile DD-LOTOS), qui étend le langage DD-LOTOS par l'introduction de l'aspect mobilité. La mobilité de processus d'un site à un autre est souvent motivée par les contraintes de performance. Dans un système réparti où les ressources sont partagées par plusieurs utilisateurs, et les charges sont variables et imprédictibles, la migration de code vers des localités moins chargées est un moyen pour exécuter efficacement des applications distribuées.

Notre modèle permet aux processus de migrer entre les sites distribués ou localités, ce qu'on appelle la mobilité des processus. Deux types de communication sont présents dans notre modèle, la communication locale, c'est à dire l'échange d'informations entre deux processus dans la même localité, et la communication à distance, c'est à dire l'échange d'informations entre deux processus de localités différentes. Cette dernière est assurée par échange de messages.

Nous présentons d'abord la syntaxe, suivie par la sémantique opérationnelle du modèle. Ensuite nous illustrons le modèle par la présentation de plusieurs exemples.

6.1 Le langage mobile DD-LOTOS

6.1.1 Discussion et motivations

Dans la modélisation de la mobilité, il faut savoir quelles sont les entités qui se déplacent, et dans quel espace se déplacent-elles ? dans[SW01], où distingue entre deux types de mobilité. Dans le premier type, c'est les liens qui se déplacent dans un espace abstrait de processus liés. Par exemple : des liens hypertextes peuvent être créés, peuvent être passés, et peuvent disparaître ; les connexions entre les téléphones cellulaires et le réseau de stations de base peuvent être changé dynamiquement. Dans le second type de mobilité, ce sont les processus qui se déplacent dans un espace abstrait de processus liés. Par exemple : le code peut migrer sur le réseau et s'exécute à son destination.

Le π-calcul considère le premier type de mobilité : il exprime directement le mouvement de liens dans l'espace des processus liés. Il y a deux sortes d'entités de base dans le π-calcul : les noms et les processus, les noms sont des noms de liens, les processus peuvent interagir en utilisant des noms qu'ils partagent. Le point crucial est que les processus peuvent échanger les noms des liens, une fois un processus reçoit un nom il peut l'utiliser pour communiquer avec les processus qui partagent ce même nom. En recevant un nom, un processus peut acquérir une capacité d'interagir avec des processus qui n'étaient pas en connexion avec lui. La topologie virtuelle du réseau de connexions entre les processus ainsi change dynamiquement au fil du temps. Le point fort du π-calcul est la façon dont il traite la portée des noms et d'extrusion des noms.

Dans le deuxième type de mobilité, ce sont les processus qui se déplacent (plus généralement, les entités composées à partir de processus), c'est le cas du π-calcul d'ordre supérieur[San92], c'est une théorie basée sur le passage de processus (process-passing)[SW01].

Le π-calcul ne mentionne pas explicitement la localité ou la distribution de processus mobiles. La question de la localisation et la distribution est orthogonale à la question du passage de nom(name-passing) ou le passage de processus (process-passing). On peut imaginer des espaces abstrait dans lesquels les processus résident dans des localités et échangent les liens.

En π-calcul, les noms sont des noms de liens. Mais c'est quoi un lien ? Le calcul n'est pas précis sur ce point : un lien est interprété de façon très large, et les noms peuvent être affectés à de nombreuses utilisations.

Dans la section suivante, nous proposons un calcul pour le calcul distribué, temps réel avec mobilité de processus. En DD-LOTOS, la distribution est assurée par la présence des localités dans le calcul.

Les applications mobiles et distribuées sont naturellement dynamiques, ce qui signifie que le nombre de processus impliqués dans le système n'est pas fixe. A tout moment il peut y avoir une nouvelle création ou une suppression processus. Pour prendre en charge l'étude de ce type d'applications, il est nécessaire d'utiliser des modèles dynamiques.

Notre calcul introduit la dynamicité en permettant au processus de migrer d'une localité à une autre. Bien que nous exigeons une nouvelle primitive pour permettre la migration

entre les localités. Ainsi nous augmentons DD-LOTOS avec la construction : $go(l, E)\{d\}$. Intuitivement, cela signifie : migrer le comportement E à la localité l, cette migration est offerte à l'environnement pour une période d unité de temps. Un autre aspect dynamique de notre calcul est la possibilité de créer de nouvelles localités, ou la suppression de localités.

Nous allons illustrer notre approche à travers un exemple représenté par le système suivant :

$$l(F \;|||\; (create(k, E) >> go(k, E')\{d\}))$$

Il est constitué d'une seule localité l. Cette localité contient deux processus parallèles, le premier F, et le second est composé par l'opérateur de composition séquentielle ($>>$), qui est composé du processus de création de la localité ($create(k, E)$) et suivie par le processus de migration ($go(k, E')\{d\}$).

Après une étape de calcul, on obtient le système :

$$l(F \;|||\; go(k, E')\{d\}) \mid k(E)$$

Cette étape consiste à créer la localité k, donc le système se décompose maintenant de deux localités l et k. Nous supposons que la migration sera activée dans l'intervalle de temps $[0, d]$, on obtient le système suivant :

$$l(F) \mid k(E'|||E)$$

Si le processus F termine son comportement, donc il se transforme en *stop*, on obtient alors :

$$l(stop) \mid k(E'|||E)$$

Dans notre calcul une localité qui contient le processus *stop*, sera supprimé, nous obtenons le système suivant :

$$k(E'|||E)$$

6.1.2 Syntaxe

Dans cette section, nous définissons la syntaxe de notre calcul de processus mobiles. Nous étendons le DD-LOTOS pour modéliser les systèmes mobiles, les processus peuvent être répartis sur plusieurs localités. Le modèle propose deux types de calcul, le calcul local sur différentes localités et la communication à distance entre les localités. Nous introduisons les outils pour prendre en compte la création de localités et de migration de comportements.

Définition 6.1 *(actions) L'ensemble des actions dans le système est comme suit :*
- L'ensemble $Act = \mathcal{G} \cup \{i, \delta, go, create\}$
- L'ensemble des actions de communication entre les localités : l'émission ou réception de

messages via les canaux de communication $Act_{com} ::= a!m \mid a?x \mid \tau$ *(action d'émission, les actions de réception et de l'action silencieuse).*

La syntaxe de notre modèle est une extension de celle de DD-LOTOS, en introduisant les aspects de mobilité.

La syntaxe du calcul est donnée par la figure 6.1. La principale catégorie syntaxique est celle des comportements E et des systèmes S. Intuitivement, un système est constitué d'un ensemble de localités composées en parallèle, les processus qui résident dans des localités différentes peuvent communiquer via les canaux de communication. Le système peut être vide ou peut contenir des comportements en exécution.

Si l'expression de comportement $go(k, E)$ est en cours dans une localité l, le comportement E peut migrer vers la localité k et ensuite commence son exécution dans cette localité. L'ensemble $\mathcal{B}$ parcouru par E, F, ... désigne toutes les expressions de comportements.

$E ::=$		**Comportements**
	...	
	$\mid go(l, E)\{d\}$	**Migration**
	$\mid create(l, E)$	**Création** de **localité**
$S ::=$		**Systèmes**
	$\phi \mid \ S \mid S \mid \ l(E)$	

FIG. 6.1 – *Syntaxe du Mobile DD-LOTOS*

Informellement la sémantique des divers constructeurs ainsi définie est la suivante :

- Le processus $go(l, E)\{d\}$ provoque la migration du comportement E vers la localité l, cette migration est offerte à l'environnement pour une période d. La migration doit être réalisée dans l'intervalle $[0, d]$, si le temps dépasse d unités de temps, l'action n'est plus offerte à l'environnement. Dans un système sans contraintes temporelles, ce processus se comporte comme suit : La localité qui contient la migration, migre vers la localité décrite dans le comportement.

- Le processus $create(l, E)$ provoque la création de la localité l, suivie de l'exécution du comportement E dans cette localité.

Exemple

Soit le système suivant :

$$l(go(k, E)\{d\} \ ||| \ F) \ \mid k(E')$$

Le système est composé de deux localités. La localité de l contient deux processus parallèles : l'action de la migration et le processus F. La localité k qui contient le processus $E\prime$.

Dans ce système, nous avons plusieurs cas :

Cas 1 :

La migration peut avoir lieu avant que le temps écoulé depuis la sensibilisation de l'action de migration ne dépasse d unités de temps, ce qui donne :

$$l(F) \mid k(E' \mid\mid\mid E)$$

Cas 2 :

Il ya une durée d', tels que $d = d' + d''$, et le temps écoulé depuis la sensibilisation de l'action de migration est d'', ce qui donne :

$$l(go(k, E)\{d'\} \mid\mid\mid F) \mid k(E')$$

Cas 3 :

Le temps écoulé dépasse la durée d unités de temps, donc la migration ne se produira jamais, ce qui donne :

$$l(F) \mid k(E')$$

6.1.3 Sémantique opérationnelle structurée :

En [MSK11], nous avons étudié deux aspects des systèmes distribués : la distribution et la communication. Dans la distribution, nous avons introduit la notion de localité, pour la communication, nous avons adopté deux types : la communication locale dans la même localité sur les portes de synchronisation et la communication à distance par échange de messages via des canaux de communication. La sémantique des opérateurs de base (stop, exit, la préemption, l'opérateur de délai, d'action avec prédicats, l'intériorisation, le choix non déterministe, la composition parallèle, la composition séquentielle) reste même que celle de D-LOTOS.

Définition 6.2 *La relation de transition de maximalité* $\rightarrow \subseteq \mathcal{C} \times Atm \cup D \times \mathcal{C}$ *est définie comme la plus petite relation satisfaisant les règles suivantes :*

- Processus de création de localités : $l(create(k, E))$

Considérons la configuration ${}_M[l(F \mid create(k, E))]$, cette configuration représente les évolutions potentielles en fonction des actions indexées par l'ensemble M. La création de localités ne peut pas se produire jusqu'à ce que les actions indexées par l'ensemble M terminent leur exécution, la condition $wait(M) = false$ dans la règle 1. Si l'action de création n'est pas sensibilisées et nous avons d unités de temps écoulées, alors on a la situation de la règle 2.

1. $$\frac{\neg wait(M)}{{}_M[l(F \mid create(k,E))] \overset{M^{create_x}}{\rightarrow} {}_\phi[l(F)] \mid_{\{x:create:0\}} [k(E)]}$$

2. $\frac{wait(M^d) \quad d>0}{{}_M[l(F \mid create(k,E))] \xrightarrow{d} {}_{M^d}[l(F \mid create(k,E))]}$

- Processus de suppression de localités **:**

Si l'ensemble des processus en cours d'exécution dans une localité est *stop*, alors on procède à la suppression de cette localité.

1. $\frac{\neg wait(M) \quad and\ [l(stop)]}{{}_M[l(stop)] \xrightarrow{\delta} {}_{\phi}[\phi]}$

2. $\frac{\neg wait(M) \quad and \quad {}_M[l(stop)] \mid\mid\mid {}_N[k(E)]}{{}_M[l(stop)] \mid\mid\mid {}_N[k(E)] \xrightarrow{\delta} {}_N[k(E)]}$

- Processus de migration : ${}_M[k(go(l,E)\{d\})]$

Considérons la configuration ${}_M[k(go(l,E)\{d\})]$, le processus de migration ne peut se produire jusqu'à ce que les actions indexées par l'ensemble M terminent leur exécution, qui est exprimée par la règle 1. La règle 2 exige que la sensibilisation de l'action *go* dans l'intervalle $[0,d]$, dans le cas contraire la migration ne sera jamais. La règle 3, exprime le passage du temps.

1. $\frac{\neg wait(M)}{{}_M[k(F \mid go(l,E)\{d\})] \xrightarrow{M^{go_x}} {}_{\phi}[k(F)] \mid_{\{x:go:0\}} [l(E)]} x = get(\mathcal{M})$

2. $\frac{\neg wait(M)\ and\ d'>d}{{}_M[k(go(l,E)\{d\})] \xrightarrow{d'} {}_M[k(Stop)]}$

3. $\frac{\neg wait(M)\ and\ d'>0}{{}_M[k(go(l,E)\{d+d'\})] \xrightarrow{d} {}_{M^d}[k(go(l,E)\{d'\})]}$

6.2 Exemples

Nous décrivons dans cette section des systèmes distribués, et dynamiques qui peuvent être modélisés dans le Mobile DD-LOTOS.

6.2.1 Création d'une nouvelle localité

Nous considérons ici la possibilité de créer une localité. Nous cherchons ainsi à générer le mobile C-DATA pour le système suivant :

$$S ::= l(E) \mid k(F)$$

Ce système ce décompose de deux localités l (resp k). La localité l comporte le comportement E décrit dans la section 4.2.3, par l'expression :

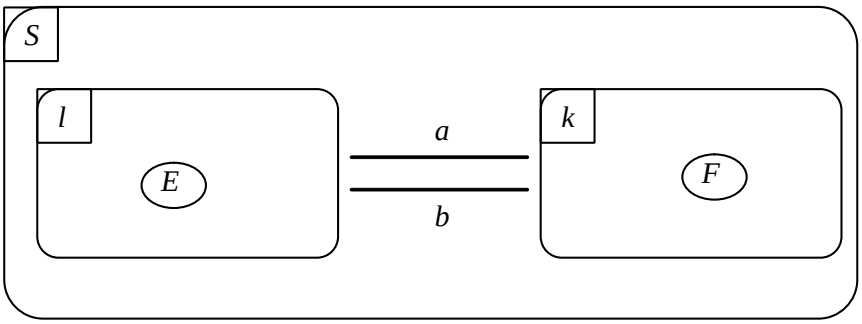

FIG. 6.2 – Système avec deux localités

$$E ::= \quad a?Rq\_Data \\ (\Delta^t(b!m\{d\}; exit))$$

La localité k comporte le comportement F suivant :

$$F ::= ((a!v; P >> b?donnee) \; [] \; create(n, Q)) >> F$$

Pour la partie communication le raisonnement de la section précédente s'applique de la même manière. La construction $create(n, Q)$ permet de créer la localité n avec le comportement Q. C'est à dire après un ensemble de pas de calcul, on aura la configuration suivante :

$$\underbrace{\phi[create(n, Q) >> F]}_{config1} \xrightarrow{\phi, create, x} \underbrace{\{x \geq d\}[F]}_{config2}$$

d : représente le temps nécessaire à la création de la localité n.

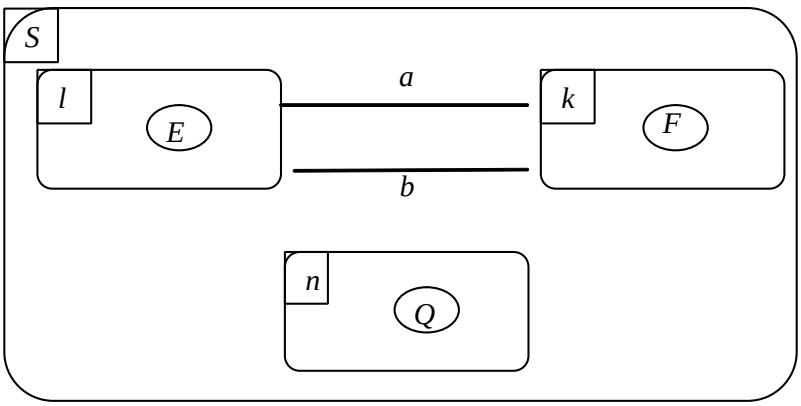

FIG. 6.3 – Création de localité

6.2.2 Migration de processus

Considérons, par exemple, un client qui accède à une base de données. Au lieu de transférer toutes les données au client et filtrer les données sur le client, le client peut envoyer le code qui fait le filtrage au serveur de base de données ; le filtrage est ainsi effectué auprès des données, et seulement le résultat est retourné au client. Le système est composé d'un client et d'un serveur illustré par la figure 6.4.

Dans cet exemple nous considérons la possibilité de migration des processus entre les localités. Nous prenons le cas le plus simple sans contraintes temporelles, c'est à dire la requête de migration n'est pas restreinte à une durée bien déterminée, comme suit :

$$l(E \;|||\; (go(k, E') >> (a?x; P))) \mid k(F)$$

Dans la localité l la requête de migration et en parallèle avec le comportement E.

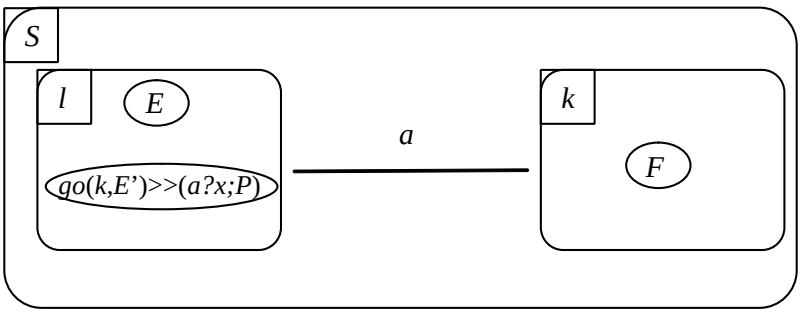

FIG. 6.4 – Migration des processus 1

La construction $go(k, E')$ permet de migrer le comportement $E\prime$ (processus de filtrage) de la localité l vers la localité k. Sur le canal a en recevant une information qui sera substituée par x, représente le résultat du processus de filtrgae. C'est à dire après un ensemble de pas de calcul, on aura la configuration suivante :

$$\underbrace{{}_{\phi}[E|||(go(k, E') >> (a?x; P))]|_{\phi}[F]}_{config1} \xrightarrow{\phi, go, x} \underbrace{{}_{\{x \geq t\}}[E|||(a?x; P)]|_{\phi}[F|||E']}_{config2}$$

t : représente le temps nécessaire à la migration du comportement E'. Cette configuration est illustrée par la figure 6.5.

Une fois le processus de filtrage termine son exécution, on aura la transition suivante :

$$\underbrace{{}_{\phi}[E|||(a?x; P)]|_{\phi}[F]}_{config3} \xrightarrow{\phi, \tau, x} \underbrace{{}_{\{x \geq d\}}[E|||P\{x/v\}]|_{\phi}[F]}_{config4}$$

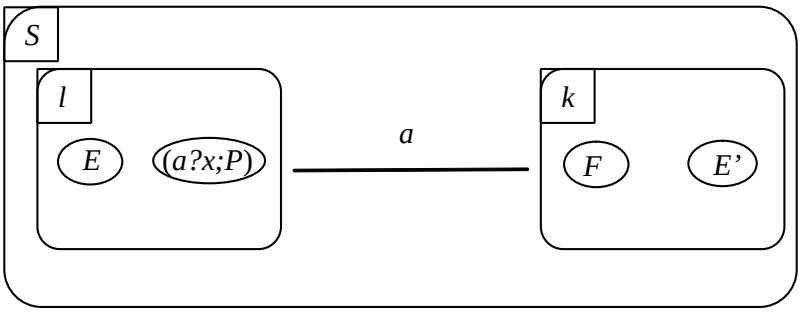

FIG. 6.5 – Migration des processus 2

6.3 Conclusion

La mobilité du code, est de plus en plus utilisée dans la programmation distribuée. Elle désigne un processus qui peut migrer sur un réseau hétérogène, et qui peut être exécuté à destination. Parmi les avantages de cette mobilité on peut citer :

* Efficacité : il est plus efficace de migrer le processus, lorsque les interactions sont répétées entre deux sites distants, afin d'interagir localement. Surtout lorsque la latence du réseau est élevée et l'interaction se compose de nombreux petits messages.

* Simplicité et flexibilité : le maintien d'un réseau peut être beaucoup plus simple lorsque les applications sont sur un serveur, et les clients, à partir du serveur téléchargent ces applications.

Dans ce chapitre nous avons proposé une extension de notre langage formel DD-LOTOS avec primitives de mobilité des processus, exprimée par la migration des entités entre les diverses localités du système, et la création, et suppression des localités. Notre langage nommé Mobile DD-LOTOS, est capable de spécifier des systèmes distribués et temps réel avec la prise en charge de la mobilité.

Chapitre 7

Conclusion et perspectives

Le travail présenté dans cette thèse, a pour objectif la spécification formelle des systèmes répartis, concurrents temps réel. Dans de tels systèmes, les contraintes temporelles sont imposées sur leurs composants distribués coopératives. L'autre caractéristique essentielle à prendre en charge dans de tels systèmes, la mobilité des processus entre les différents composants distribués.

Dans la première partie, nous avons présenté un survol des modèles et calculs sur lesquelles nous nous sommes inspirés pour les propositions que nous avons développés. Dans le contexte de la modélisation de processus concurrents, plusieurs calculs et algèbres de processus ont été introduits parmi lesquelles : le π-caluclus[MPW92], introduit par R. Milner. Dans ce modèle la mobilité, se traduit par l'existence de noms, qui sont indissociables des processus de communication, chose fondamentale dans tout système concurrent distribué. L'existence de noms suggère également un espace abstrait du processus relié, dans lequel les noms représentent les connexions. Seuls les processus qui partagent les noms sont capables d'interagir. La structure du système change donc de manière dynamique, car les liens entre les processus sont sans cesse créés et détruits. Le $\pi_{1l} - calculus$[Ama97][Ama00] asynchrone, étendu avec une notion de localité et un modèle de défaillances. Ce modèle dispose d'une notion explicite du site ou localité. Dans l'aspect mobilité, les entités migrantes sont les processus. Le $D\pi - calculus$[HR02], une extension mineure du $\pi - calcul$. Les entités de base dans $D\pi$ sont des processus et systèmes. Un système est un ensemble de processus en parallèle. Il n'y a pas de communication à distance : un message pour lequel le récepteur correspondant est dans un domaine éloigné doit migrer vers sa destination. Les processus sont des entités migrantes. $Djoin - calculus$[FGJ+96][Fou98], est un modèle pour la programmation mobile qui inclut des fonctionnalités explicites pour gérer les localités et un modèle de défaillance. Les entités de base de Djoin-calcul sont des messages, des définitions, processus, pattern de message et de solutions. En Djoin-calcul les localités ne peuvent pas migrer. Inspiré du π-caluclus, le calcul des Ambients, considère que les processus sont exécutés dans un environnement hiérarchique imbriquée appelée ambient et qui peuvent migrer d'un ambient à un autre. Un ambient est un endroit qui est délimité par des limites. Chaque ambient, contient

un ensemble de processus locaux, et des sous-ambients, il peut se déplacer dans un autre ambients ou quitter un ambient, par ces capacités qui sont associées aux noms des ambients. La communication et l'interaction entre les différents domaines peuvent être réalisées grâce à la migration des ambients et la capacité d'ouverture. M-calculus[SS03], calcul d'ordre supérieur représente une extension de Djoin-calcul avec les localités programmables. le M-calcul retient l'idée de communication asynchrone, avec des définitions avec des pattern de messages pour la synchronisation. Les entités de base pour la distribution sont les localités. La migration est réalisée grâce de l'ordre supérieur de la communication. Le langage KLAIM[RGR98], extension du langage LINDA, avec les notions de sites (emplacements physiques), les localités (emplacements logique) et la migration des agents. Le mécanisme de communication est asynchrone et est basé sur le concept d'espace tuple. Les entités mobiles sont des processus. Il y a deux types de migration (deux instructions). Le premier type migre un processus à la localité avec le contexte (contexte mobile), et le deuxième type migre un processus à une localité où il sera exécutée dans le contexte de cette localité. PICT[Pie98], extension basée sur le π-calculus asynchrone. NomadicPICT[Uny02], extension de PICT avec la notion de localité et migration des processus. Les entités de base sont les processus et les agents. NomadicPICT propose deux types d'entités mobiles : la migration des processus et la migration des messages.

Ensuite nous nous sommes intéressés aux calculs temps réel, à savoir le langage RT-LOTOS et D-LOTOS : deux extensions de l'algèbre de processus LOTOS. Notre choix porte sur le langage D-LOTOS, ce choix est motivé par la capacité à spécifier les contraintes temporelles et les durées des actions. Cette dernière est une caractéristique essentielle pour l'analyse des performances des systèmes temps réel et permet essentiellement d'évaluer si le système répond à ses contraintes de temps. Dans ce contexte la sémantique de maximalité a fait montrer[SC03b], sa puissance pour prendre en charge de manière explicite les durées des actions.

Dans la partie contribution nous avons mis en évidence les difficultés associés au modèle DATA*, pour prendre en charge les deux aspects : distribution du calcul et mobilité des systèmes concurrents, pour répondre à ces insuffisances nous avons introduit de manière formelle deux modèles sémantiques dans l'esprit des systèmes de transitions étiquetés : Communicating DATA (C-DATA) et mobile C-DATA. Le modèle C-DATA extension du modèle DATA*, pour prendre en charge la communication par échange de messages via des canaux de communication entre les différentes localités du calcul. Le modèle mobile C-DATA extension de C-DATA dont laquelle les processus peuvent migrer d'une localité à une autre, en plus les localités dans le calcul peuvent évoluer dans le temps, c'est à dire, ils disposent d'une structure dynamique. Dans la dernière section de ce chapitre nous nous sommes exploité la large utilisation des automates temporisés pour la vérification des systèmes temps réel, par la proposition d'un algorithme de transformation des DATA* en automates temporisés. L'objectif de cette transformation est la vérification formelle et quantitative des propriétés telles que le respect des contraintes temporelles et performances minimales, en exploitant les

automates temporisés. Ce principe, permet de bénéficier des outils et des résultats obtenus sur les automates temporisés.

Ensuite nous avons définit un langage nommée DD-LOTOS, une extension de D-LOTOS. Une spécification DD-LOTOS est traduite vers le modèle C-DATA pour une éventuelle vérification. Nous avons aussi définit le langage "Mobile DD-LOTOS".

Le travail présenté dans cette thèse nous ouvre des perspectives intéressantes et nous comptons prolonger ce travail dans plusieurs directions. Tout d'abord, nous allons réaliser une implémentation du calcul, en nous basant sur les techniques utilisées pour implémenter Join calcul. Développer une théorie algébrique pour le calcul avec des relations de bisimulations. Comparer, notre langage par rapport à d'autres langages ou calculs, par exemple Djoin-calcul ou Dπcalcul, en terme de puissance d'expression en utilisant les techniques dites *Embeddings*[BP90][BP94][Sha92].

Dans le volet vérification formelle, proposition d'une logique temporelle adéquate à la spécification des propriétés quantitatives pour le calcul des performances, ensuite développement d'un modèle checker qui permet de vérifier ce type de propriétés sur le modèle mobile C-DATA.

Bibliographie

[ABL99] Roberto M. Amadio, Gérard Boudol, and Cédric Lhoussaine. The receptive distributed pi-calculus (extended abstract). In *Proceedings of the 19th Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 304–315, 1999.

[AC98] Roberto M. Amadio and Pierre-Louis Curien. *Domains and Lambda-calculi.* Cambridge University Press, 1998.

[ACS96] Roberto M. Amadio, Ilaria Castellani, and David Sangiorgi. On bisimulations for the asynchronous pi-calclulus. In U. Montanari and V. Sassone, editors, *Proceedings of CONCUR'96*, volume 1119 of *LNCS*, pages 147–162. Springer-Verlag, 1996.

[AD94] R. Alur and D. L. Dill. A theory of timed automata. *TCS*, 126 :183–235, 1994.

[AFH94] R. Alur, L. Fix, and T. A Henzinger. Event-clock automata : A determinizable class of timed automata. In *Proc. 6th International Conference on Computer Aided Verification (CAV'94)*, volume 818 of *LNCS*, pages 1–13. Springer-Verlag, 1994.

[Alo41] Church Alonzo. *The calculi of Lambda-Conversion.* Princeton University Press, 1941.

[Ama97] Roberto M. Amadio. An asynchronous model of locality, failure, and process mobility. In *Proceedings Second International Conference on Coordination Models and Languages COORDINATION 97*, volume 1282 of *LNCS*, pages 374–391. Springer-Verlag, 1997.

[Ama00] Roberto M. Amadio. On modelling mobility. *Theoretical Computer Science*, 240 :147–176, 2000.

[AP94] Roberto M. Amadio and Sanjiva Prasad. Localities and failures. In *Proc. of FST TCS*, volume 880, pages 205–216. Springer LNCS, 1994.

[Bae04] J. C. M. Baeten. A brief history of process algebra. Technical report, Theor. Comput. Sci, 2004.

[BB91] J. C. M. Baeten and J. A. Bergstra. Real time process algebra. *Formal Aspects of Computing*, 3 :142–188, 1991.

[BB92] Gérard Berry and Gérard Boudol. The chemical abstract machine. *Theoretical Computer Science*, 96 :217–248, 1992.

[BCGL02] Gérard Boudol, Ilaria Castellani, Florence Germain, and Marc Lacoste. Models of distribution and mobility : State of the art. Research report D1.1.1, MIKADO Project, 2002.

[BGL00] Gérard Boudol, Florence Germain, and Marc Lacoste. Analyse des langages et modèles de la mobilité. Research reportINRIA RR-3930, INRIA Sophia-Antipolis, 2000.

[BK84] J. A. Bergstra and J. W. Klop. Process algebra for synchronous communication. *Information and control*, 60 :109–137, 1984.

[BL91] Tommaso Bolognesi and Ferdinando Lucidi. Lotos-like process algebras with urgent or timed interactions. In Ken R. Parker and Gordon A. Rose, editors, *FORTE*, volume C-2 of *IFIP Transactions*, pages 249–264. North-Holland, 1991.

[Bou92] Gérard Boudol. Asynchrony and the pi-calculus. Technical report 1702, INRIA, Sophia-Anpolis, 1992.

[BP90] F.S. De Boer and C. Palamidessi. Concurrent logic programming : Asynchronism and langauage comparison. In S. Debray and H. Hermenegildo, editors, *Proc of 1990 North American Conference on Logic Programming*, pages 175–194. MIT Press, 1990.

[BP94] F.S. De Boer and C. Palamidessi. Embeddings as a tool for langauage comparison. *Information and Computation*, 115 :128–157, 1994.

[BS05] N. Belala and D. E. Saïdouni. Non-Atomicity in Timed Models. In *International Arab Conference on Information Technology (ACIT'2005)*, Al-Isra Private University, Jordan, December 2005.

[Car95] L. Cardelli. A language with distributed scope. *Computing Systems*, 8(1) :27–59, 1995.

[Car97] L. Cardelli. Mobile ambient synchronization. *Digital systems Research Center*, July 1997.

[Cas01] Ilaria Castellani. Process algebra with localities. In SCOTT SMOLKA JAN BERGASTRA, ALBAN PONSE, editor, *Handbook of Process Algebra*, pages 945–1045. Elsevier, 2001.

[CdO95a] J. P. Courtiat and R.C. de Oliveira. On RT-LOTOS and its application to the formal design of multimedia protocols. *Annals of Telecommunications*, 50 :11–12, 1995.

[CdO95b] J. P. Courtiat and R.C. de Oliveira. A reachability analysis of RT-LOTOS specifications. In *FORTE*, London, 1995. Chapman and Hall.

[CdS93a] J. P. Courtiat, M. S. de Camargo, and D. E. Saïdouni. RT-LOTOS : LOTOS temporisé pour la spécification de systèmes temps réel. In R. Dssouli, G.V. Bochmann, and L. Levesque, editors, *Ingénierie des Protocoles (CFIP'93)*, pages 427–441. Hermes, 1993.

[CdS93b] J. P. Courtiat, M. S. de Camargo, and D. E. Saïdouni. RT-LOTOS : LOTOS temporisé pour la spécification de systmes temps rél. Research Report 93040, LAAS-CNRS, 7 av. du Colonel Roche, 31077 Toulouse Cedex France, February 1993.

[CES86] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2) :244–263, April 1986.

[CG98] L. Cardelli and A. D. Gordon. Mobile ambients. *Lecture Notes in Computer Science*, 1378 :140–155, 1998.

[CG99] L. Cardelli and A . D. Gordon. Types for mobile ambients. *POPL*, pages 79–92, January 1999.

[CSLO00] J. P. Courtiat, C . A. S. Santos, C. Lohr, and B. Outtaj. Experience with RT-LOTOS, a temporal extension of the LOTOS formal description technique. *Computer Communications*, 23 :1104–1123, 2000.

[EM85] H. Ehrig and B. Mahr. *Fundamentals of Algebraic Specification 1 : Equations and Initial Semantics*. Number vol. 1 in Berlin TU. Technical University of Berlin, 1985.

[Ene01] Cristian Ene. *Un modèle formel pour les systèmes mobile à diffusion*. PhD thesis, Université d'Aix-Marseille II, 2001.

[FDI01] ISO/IEC FDIS. *Information technology – Enhancements to LOTOS*. International Organisation of Standardization – Information Processing Systems – Open Systems Interconnection, Genève, 2001.

[FGJ+96] C. Fournet, G.Gonthier, J.J.Lévy, L.Maranget, and D.Rémy. A calculus of mobile agents. *Montanari and Sassone*, 103 :406–421, 1996.

[Fou98] C. Fournet. *A calculus for Distributed Mobile Programming.* PhD thesis, Ecole Polytechnique, Palaiseau France, 1998.

[Gen85] David Genernter. Generative communication in linda. *ACM Transactions on Programming Language and Systems*, 7(1) :80–112, 1985.

[GNS00] Florence Germain, Elie Najm, and Jean-Bernard Stefani. Elements of an objects-based model for distributed and mobile computation. In *Fourth International Conference on Formal Methods for Open Object-Based Distributed Systems (FMOODS'00)*, Palo Alto, California, 2000. Kluwer.

[GP06] G. Giobanu and C. Prisacariu. Timers for distributed systems. *Electronic Notes in Theoretical Computer Science*, pages 81–99, 2006.

[Hen07] Matthew Hennessy. *A Distributed Pi-Calculus.* Cambridge University Press, 2007.

[HNSY92] T. A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. In *7th. Symposium of Logics in Computer Science*, pages 394–406, Santa-Cruz, California, 1992. IEEE Computer Scienty Press.

[Hoa78] C. A. R. Hoare. Communicating sequential processes. *Communications of ACM*, 21(8) :666–677, August 1978.

[Hoa85] C. A. R. Hoare. *Communicating Sequential Processes.* Prentice-Hall, Englewood Cliffs, New Jersey, 1985.

[HR02] M. Hennessy and J. Riely. Resource access control in systems of mobile agents. *Information and Computation*, 173 :82–120, 2002.

[HT91] K. Honda and M. Tokoro. An object calculus for asynchronous communication. In Pierre America, editor, *European Conference on Object-Oriented Programming (ECOOP'91)*, volume 512 of *LNCS*, pages 133–147, Berlin, 1991. Spring-Verlag.

[HT92] K. Honda and M. Tokoro. On asynchronous communication semantics. In Mario Tokoro, Oscar Nierstrasz, and Peter Wegner, editors, *European Conference on Object-Oriented Programming (ECOOP'92)*, pages 21–51. Spring-Verlag, 1992.

[ISO88a] ISO8807. *LOTOS, A Formal Description Technique Based on the Ordering of Observational Behaviour.* ISO, November 1988.

[ISO88b] ISO/IEC. *LOTOS – A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour, International Standard 8807.* International Organisation of Standardization – Information Processing Systems – Open Systems Interconnection, Genève, September 1988.

[JM04] O. H. Jensen and R. Milner. Bigraphs and mobile processes(revised). Tech. report, University of Cambridge, Computer Laboratory, 2004.

[KC08a] L. Kahloul and A. Chaoui. Code mobility modeling : a temporal labelled reconfigurable nets. In *Proceedings of the 1st International Conference on MOBILe Wireless MiddleWARE, Operating Systems, and Applications, MOBILWARE 2008, ACM International Conference Proceeding Series 278*, Innsbruck,Austria, February 2008.

[KC08b] L. Kahloul and A. Chaoui. Coloured reconfigurable nets for code mobility modeling. *International Journal of Computers, Communications Control*, 3 :358–363, 2008.

[KC10] L. Kahloul and A. Chaoui. Modeling reconfirgurable systems using flexible petri nets. In *4th IEEE International Symposium on Theoretical Aspects of Software Engineering*, pages 24–27, Taipei, Taiwan, August 2010.

[Kop11] Hermann Kopetz. *Real-Time Systems : Design principles for distributed embedded applications*. Springer-Verlag, 2011.

[LDBY03] K. G. Larsen, A. David, G. Behrmann, and W. Yi. A tool architecture for the next generation of uppaal. Technical report, Uppsala University, 2003.

[Led91] G. Leduc. An upward compatible timed extension to LOTOS. In *FORTE*, pages 223–238. North Holland, 1991.

[LL97] L. Léonard and G. Leduc. An Introduction to ET-LOTOS for the Description of Time-Sensitive Systems. *Computer Networks and ISDN Systems*, 29 :271–292, 1997.

[LL98] L. Léonard and G. Leduc. A formal definition of time in LOTOS - extended abstract. *Formal Aspects of Computing*, 10(3) :248–266, 1998.

[Loh02] C. Lohr. *Contribution à la Conception de Systèmes Temps-Réel S'appuyant sur la Technique de Description Formelle RT-LOTOS*. PhD thesis, LAAS-CNRS, 7 avenue du colonel Roche, 31077 Toulouse Cedex France, 2002.

[LPY97] K. G. Larsen, P. Pettersson, and W. Yi. UPPAAL in a nutshell. *Journal of Software Tools for Technology Transfer (STTT)*, 1(1-2) :134–152, 1997.

[Mil80] R. Milner. *A Calculus of Communicating Systems*, volume 92 of *LNCS*. Springer-Verlag, 1980.

[Mil83] R. Milner. Calculus for synchrony and asynchrony. *TCS*, 25 :267–310, 1983.

[Mil89] R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[Mil90] R. Milner. Functions as processes. In *ICALP : Annual International Colloquium on Automata,*. Languages and programming, 1990.

[Mil93] R. Milner. The polyadic pi-calcul : a tutorial. In *Logic and Algebra of Specification*, University of Edinburgh UK, 1993. Spring-Verlag.

[Mil99] R. Milner. *Communicating and Mobile Systems : The pi-Calculus*. Cambridge University Press, May 1999.

[Mil09] Robin Milner. *The Space and Motion of Communicating Agents*. Cambridge University Press, 2009.

[MPW92] R. Milner, Joahim Parrow, and David Walker. A calculus of mobile processes,Parts I and II. *Information and Compution*, pages 100–140,41–77, September 1992.

[MS09] T. M. Maarouk and D. E. Saïdouni. Prise en charge de la mobilité dans les algèbres de processus temps réel. In *Proceedings of International Conference on Applied Informatics ICAI'2009*, pages 10–15, Bordj Bou Arréridj, Algeria, 15-17, 2009.

[MS11] T. M. Maarouk and D. E. Saïdouni. Formal verification of real-time systems using a true concurrency semantics. In *2nd International Conference on Information and Multimedia Technology ICIMT'2011.*, Dubai, UAE, 2011.

[MSB12] T. M. Maarouk, D. E. Saïdouni, and A. Bezza. Environment of specification of real-time systems. In *Proceedings International Conference on Information Systems and Technology ICIST'2012*, pages 99–103, Sousse, Tunisie, 2012.

[MSK11] T. M. Maarouk, D. E. Saïdouni, and M. Khergag. DD-LOTOS : A distributed real time language. In *Proceedings 2nd Annual International Conference on Advances in Distributed and Parallel Computing (ADPC 2011) Special Track : Real Time Embedded Systems (RTES 2011)*, pages 45–50, Singapore, 2011.

[MSK12] T. M. Maarouk, D. E. Saïdouni, and M. Khergag. Towards a calculus for distributed, real-time and mobile systems. *Journal of Software(JSW, ISSN 1796-217X)*, 7(3) :564–574, 2012.

[MT90] F. Moller and C. Tofts. A temporal calculus of communicating systems. In J. C. Baeten and J. W. Klop, editors, *CONCUR*, volume 458 of *LNCS*, pages 401–415. Springer-Verlag, 1990.

[OP92] F. Orava and J. Parrow. An algebraic verification of a mobile network. *Journal of Formal Aspects of Computing*, 4 :497–543, 1992.

[Pal03] C. Palamidessi. Comparing the expressive power of the synchronous and the asynchronous pi-calculus. *Mathematical Structures in Computer Science*, 13(5) :685–719, 2003.

[Pie98] B. C. Pierce. Programming in the Pi-Calculus : A Tutorial Introduction to Pict (Pict version 4.1). Technical report, Indiana University, 1998.

[Pra95] K. V. S. Prasad. A calculus of broadcasting systems. *Science of computer Programming*, 25, 1995.

[PV98] Joachim Parrow and Bjorn Victor. The Fusion Calculus : Expressing and symmetry in mobile process. In *13th Annual IEEE Symposium on Logic in Computer Science(LICS'98)*, 1998.

[RGR98] De Nicola Rocco, Ferrari GianLuigi, and Pugliese Rosario. KLAIM : A Kernel Lanaguage for Agents Interaction and Mobility. *IEEE Transactions on Software Engineering*, 24(5) :315–330, 1998.

[RGR00] De Nicola Rocco, Ferrari GianLuigi, and Pugliese Rosario. Programming Access Control : The Klaim Experience. In *Proc. of the 11th International Conference on Concurrency Theory (CONCUR'00)*, volume 1877 of *LNCS*, pages 48–65. Springer, 2000.

[RGRV00] De Nicola Rocco, Ferrari GianLuigi, Pugliese Rosario, and B. Venneri. Types for aceess control. *Theoretical Computer Science*, 240(1) :215–254, 2000.

[RR86] George M. Reed and A. W. Roscoe. A timed model for communicating sequential processes. In *ICALP*, pages 314–323, 1986.

[Saï96] D. E. Saïdouni. *Sémantique de Maximalité : Application au Raffinement d'Actions en LOTOS*. PhD thesis, LAAS-CNRS, 7 av. du Colonel Roche, 31077 Toulouse Cedex France, 1996.

[San92] D. Sangiorgi. *Expressing Mobility in Process Algebras : First-Order and Higher Order Paradigm*. PhD thesis, University of Edinburgh, 1992.

[San97] D. Sangiorgi. The name discipline of uniform receptiveness. In *Proc. International Colloquium on Automata, Languages and Programming (ICALP'97)*, volume 1256 of *Lecture Notes in Computer Science*, pages 303–313. Springer-Verlag, 1997.

[SB06] D. E. Saïdouni and N. Belala. Actions duration in timed models. In *International Arab Conference on Information Technology (ACIT'2005)*, Yarmouk University, Irbid, Jordan, December 2006.

[SBG11] Djamel Eddine Saïdouni, Amel Boumaza, and Souad Guellati. Prise en compte des durées d'actions dans la vérification des automates temporisés. In *1st International Conference on Information Systems and Technologies ICIST'2011*, Algeria, Tebessa University, April 2011.

[SC03a] D. E. Saïdouni and J. P. Courtiat. Prise en compte des durées d'action dans les algèbres de processus par l'utilisation de la sémantique de maximalité. In *Ingénierie Des Protocoles (CFIP'2003)*. Hermes, France, 2003.

[SC03b] D. E. Saïdouni and J. P. Courtiat. Prise en compte des durées d'action dans les algèbres de processus par l'utilisation de la sémantique de maximalité (version étendue). Technical Report 03243, LAAS-CNRS, 7 avenue du colonel Roche, 31077, Toulouse Cedex France, May 2003.

[Sch93] Werner Schutz. *The testability of distributed real-time systems*. Kluwer Academic, 1993.

[Sha92] E. Y. Shapiro. Embeddings among concurrent programming languages. *Lecture Notes in Computer Science*, 630 :486–503, 1992.

[SS03] Alan Schmitt and Jean-Bernard Stefani. The M-Calculus : A Higher Order Distributed Process Calculus. In *Proceeding 30th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2003)*, New Orleans, LA, USA, January 2003.

[SW01] D. Sangiorgi and D. Walker. *The pi-Calculus : A theory of mobile processes*. Cambridge University Press, 2001.

[TvS02] Andrew S. Tanenbaum and Maarten van Steen. *Distributed Systems : Principales and Paradigms*. Prentice-Hall, Englewood cliffs, New Jersey, 2002.

[Uny02] A. Unyapoth. *Nomadic Pi Calculi : Expressing and Verifying Infrastructure for Mobile Computation*. PhD thesis, University of Cambridge, Computer Laboratory, 2002.

[Vic98] B. Victor. *The Fusion Calculus : Expressing and Symmetry in Mobile Process*. PhD thesis, Department of Computer Systems Uppsala University, Sweeden, 1998.

[Yov97] S. Yovine. KRONOS : A verification tool for real-time systems. *Journal of Software Tools for Technology Transfer*, 1(1-2) :123–133, October 1997.