République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université A. Mira de Béjaïa
Faculté des Sciences Exactes
Département d'Informatique

Mémoire de Fin d'Étude

En vue de l'obtention du diplôme de Master en Informatique
Option : Systèmes d'information avancés

Thème :

Framework théorique pour l'analyse des systèmes cyber-physiques – Use Case : Cycab

---

Réalisé par : M. AMZAL Kamal
M. AMRANE M'hand

Encadré par : Mme CHABANE Sarah

Devant le jury composé de :

Président :	Mme BOUKERRAM Samira	Université de Béjaïa
Examinateur :	M. AMROUN Kamal	Université de Béjaïa
Examinateur :	Mme ADEL Karima	Université de Béjaïa
Examinateur :	M. KHAMMARI Mohammed	Université de Béjaïa

Promotion : 2024 – 2025

Remerciements

Au terme de ce mémoire, nous souhaitons exprimer nos plus sincères remerciements à toutes les personnes qui ont contribué, de près ou de loin, à la réalisation de ce travail.

Nous remercions avant tout **Dieu le Tout-Puissant**, pour nous avoir accordé la santé, la patience et la persévérance nécessaires pour mener à bien ce projet académique.

Nos remerciements les plus respectueux s'adressent à notre encadrante, Mme **CHABANE Sarah**, pour sa disponibilité, ses conseils pertinents, son exigence scientifique, et son encadrement bienveillant tout au long de ce travail. Son accompagnement rigoureux et ses orientations claires ont été essentiels dans la progression de notre recherche.

Nous exprimons également notre profonde gratitude aux membres du jury : Mme **Boukerram Samira**, M. **Amroun Kamal**, M. **Khammari Mohammed** et Mme **Adel Karima**, pour avoir accepté d'évaluer notre travail. Leurs remarques et suggestions sont précieuses pour l'amélioration continue de nos compétences et de nos travaux futurs.

Nos pensées les plus affectueuses vont à **nos parents**, pour leur soutien moral, leur encouragement quotidien, et leur confiance inconditionnelle. Leur présence et leur amour ont été des piliers précieux dans toutes les étapes de notre parcours

Enfin, nous tenons à remercier chaleureusement tous les membres du département d'Informatique de l'Université de Béjaïa, pour leur soutien et leur contribution à notre formation.

Dédicaces

Je dédie ce mémoire à ceux qui ont illuminé mon chemin par leur amour, leur patience et leur soutien inconditionnel.

À mes parents bien-aimés, que nulle parole ne saurait suffire à remercier pour leur immense sacrifice, leur patience inépuisable, leur présence constante et leur foi en moi. Ce travail est le fruit de leur soutien et de leur amour.

À mon frère : Faiz

À mes sœurs : Zina, Yasmine et **Ania**,

À ma grand-mère : Fatma

À ma tante : Baya.

À mon binôme AMZAL Kamal, pour sa rigueur, sa collaboration exemplaire et son engagement constant tout au long de ce travail partagé.

À mon encadrante, **Mme CHABANE Sarah**, pour sa rigueur scientifique, son accompagnement bienveillant et ses précieux conseils tout au long de ce travail.

Et enfin, **à tous ceux qui me connaissent de près ou de loin**, et qui ont contribué par un mot, un geste ou une pensée à la réalisation de ce mémoire. À chacun d'eux, j'adresse toute ma reconnaissance.

M'hand

Dédicaces

Je dédie ce mémoire à ceux qui ont illuminé mon chemin par leur amour, leur patience et leur soutien inconditionnel.

À la mémoire de mon père, dont l'absence physique n'a jamais altéré la force de sa présence dans mon cœur. Que ce travail soit un humble hommage à sa bienveillance, à ses sacrifices et à son amour inconditionnel.

À ma chère mère, pour son amour, sa patience, ses prières silencieuses et son soutien indéfectible dans chaque étape de ma vie.

À mes frères : Juba et Idir,

À ma sœur : Malha,

À mon oncle : Aimad,

À mon binôme M'hand Amrane, pour sa rigueur, sa collaboration exemplaire et son engagement constant tout au long de ce travail partagé.

À notre encadrante, Mme CHABANE Sarah, pour sa disponibilité, sa bienveillance et ses orientations précieuses, qui ont donné un sens clair à notre démarche.

Et enfin, **à tous ceux qui me connaissent de près ou de loin**, et qui ont contribué par un mot, un geste ou une pensée à la réalisation de ce mémoire. À chacun d'eux, j'adresse toute ma reconnaissance.

Kamal

Résumé

Les systèmes cyber-physiques (CPS) constituent une évolution majeure des infrastructures modernes, combinant des composants numériques et physiques pour interagir en temps réel. Ces systèmes sont omniprésents dans des domaines critiques tels que l'industrie, la santé et les transports, où la fiabilité et les performances sont essentielles. Cependant, leur complexité croissante soulève des défis importants, notamment la gestion de la modularité, la sécurité, la synchronisation et la correction par construction.

Ce mémoire propose un cadre théorique pour l'analyse et la conception des systèmes réactifs dans les CPS, basé sur les modèles synchrones tels que les I/O Automata . Nous étudions leurs avantages, leurs limites et leurs applications dans des environnements complexes. Notre objectif principal est de présenter un modèle réactif et d'évaluer ses performances à travers des tests et des simulations rigoureuses.

Notre contribution repose sur la proposition d'un nouvel opérateur de composition parallèle dédié aux systèmes réactifs synchrones présents dans les CPS. Cet opérateur enrichit la sémantique d'un modèle existant et garantit la construction correcte de systèmes complexes, et fournit un cadre théorique pour la vérification et la validation formelle.

Abstract

Cyber-Physical Systems (CPS) represent a major advancement in modern infrastructures, combining digital and physical components to interact in real-time. These systems are ubiquitous in critical domains such as industry, healthcare, and transportation, where reliability and performance are essential. However, their increasing complexity raises significant challenges, including modularity management, security, synchronization, and correctness by construction.

This thesis proposes a theoretical framework for the analysis and design of reactive systems within CPS, based on synchronous models such as I/O Automata. We explore their advantages, limitations, and applications in complex environments. Our main objective is to present a reactive model and evaluate its performance through rigorous testing and simulation.

Our contribution lies in the proposal of a new parallel composition operator dedicated to synchronous reactive systems found in CPS. This operator enhances the semantics of an existing model and ensures the correct construction of complex systems. and provides a theoretical foundation for formal verification and validation.

Table des matières

Table des figures

Liste des tableaux

Liste des Abréviations

CPS	Systèmes Cyber-Physiques (Cyber-Physical Systems)
IoT	Internet des Objets (Internet of Things)
I/O Automata	Automates d'Entrée/Sortie (Input/Output Automata)
KPN	Réseaux de Kahn (Kahn Process Networks)
SDL	Langage de Description et de Spécification (Specification and Description Language)
SR-modèles	Modèles Synchrones Réactifs (Synchronous-Reactive models)
SCADE	Environnement de Développement d'Applications Critiques (Safety Critical Application Development Environment)
BIP	Comportement, Interaction, Priorité (Behavior, Interaction, Priority)

Introduction Générale

Les systèmes cyber-physiques (CPS) sont devenus des éléments fondamentaux dans les infrastructures modernes, en raison de leur capacité à intégrer des composants numériques et physiques interagissant en temps réel. Ils sont présents dans des domaines aussi variés que l'industrie, la santé, les transports et l'énergie, où leur fiabilité et leur performance sont cruciales.

Cependant, la complexité croissante des CPS soulève plusieurs défis majeurs, notamment la gestion de la modularité, la sécurité, la compatibilité des systèmes et assurer la correction par construction de systèmes complexes. Face à ces enjeux, la tendance est à la réutilisation de composants déjà développés et testés, la réutilisation diminue considérablement le coût et le temps de production, mais relève de nouveau défi comme la fiabilité des systèmes obtenus.
L'approche de conception par composition accélère le développement en assemblant des composants indépendants pour construire des systèmes complexes. L'approche orientée composants privilégie la réutilisation du code via une standardisation stricte : chaque composant possède une spécification définissant ses services requis/fournis et son comportement, garantissant la compatibilité entre composants. Seuls les composants compatibles peuvent être combinés, assurant la correction du système par construction. Ce mémoire s'inscrit dans le contexte du développement de systèmes fiables par construction,une approche modulaire compositionnelle, qui prône l'utilisation des méthodes formelles pour garantir la conception de systèmes corrects par construction.

Dans ce travail, nous nous intéressons à la modélisation formelle des systèmes réactifs, notamment à leur construction modulaire à partir de composants. Les SR-modèles, fondés sur les automates d'entrée/sortie [35], constituent un cadre formel efficace pour représenter le comportement de tels systèmes. Toutefois, ces modèles existants ne prennent en charge qu'une configuration spécifique : la composition en série des composants.

Cette restriction limite leur capacité à modéliser des architectures plus variées, telles

que celles comportant des interactions parallèles entre composants. Pour remédier à cela, nous proposons une extension des SR-modèles permettant la composition parallèle. Cette contribution a pour objectif de rendre le cadre plus générique, en permettant de représenter des systèmes réactifs complexes, construits par assemblage de modules indépendants, tout en garantissant la cohérence du comportement global.

Pour cela, le mémoire est structuré en quatre chapitres :

— **Chapitre 1 :** expose les fondements des systèmes réactifs en détaillant leurs caractéristiques distinctives et leur intégration dans les systèmes cyber-physiques (CPS).

— **Chapitre 2 :** se concentre sur la modélisation des systèmes réactifs, en accordant une attention particulière aux modèles synchrones et aux automates d'entrée/sortie (I/O Automata).

— **Chapitre 3 :** constitue le cœur de notre contribution en présentant un modèle étendu spécifiquement conçu pour les systèmes réactifs.

— **Chapitre 4 :** offre un aperçu pratique de l'implémentation de notre modèle, illustrée par une évaluation sur une étude de cas concrète : le véhicule autonome CyCab.

Ce travail vise ainsi à poser les bases théoriques d'une méthodologie robuste pour la conception et l'analyse comportementale des CPS, tout en assurant leur fiabilité.

Chapitre 1

Généralités sur les systèmes réactifs

Introduction

Dans la conception des systèmes modernes, la nécessité de garantir une réactivité optimale et une gestion efficace des interactions est devenue une priorité. Avec l'évolution des technologies embarquées et distribuées, les défis liés à la fiabilité, à la scalabilité et à l'interopérabilité des systèmes se sont multipliés. Ces défis font l'objet de plusieurs études qui s'intérèssent aux architectures capables de gérer des événements en temps réel, en assurant un équilibre entre performance et fiabilité. En mettant en avant les approches de modélisation et d'optimisation, ces approches offrent une vision approfondie des mécanismes garantissant la correction par construction. Dans ce travail, nous nous intéressons aux systèmes réactifs dans des environnements complexes. Pour cela, nous allons aborder dans ce chapitre les différentes catégories de systèmes, leurs forces et faiblesses, ce qui nous permettra de faire une analyse approfondie afin d'avoir une base solide pour la suite de notre travail, et proposer une solution adaptée aux besoins des systèmes Cyber-Physiques.

1.1 Les Systèmes Cyber-Physiques (CPS)

1.1.1 Définition CPS

Les systèmes cyber-physiques (CPS) représentent une nouvelle génération de systèmes qui allient des capacités de calcul et des éléments physiques, interagissant de manière étroite avec leur environnement. D'après Baheti et Gill [4], les CPS se distinguent par leur capacité à fusionner le calcul, la communication et le contrôle pour interagir avec le monde physique à travers diverses modalités, souvent en temps réel. Ces systèmes

vont au-delà des méthodes traditionnelles des systèmes embarqués en mettant l'accent sur une intégration synergique des composants logiciels et matériels, ainsi que sur leur faculté à s'adapter de manière dynamique aux conditions changeantes.

La recherche en CPS vise à intégrer des principes issus de disciplines variées, notamment l'ingénierie, l'informatique et les sciences physiques, afin de développer des outils et méthodes pour concevoir des systèmes fiables et performants. Les défis incluent la vérification et validation de composants complexes, l'intégration efficace de sous-systèmes indépendants, ainsi que le développement d'architectures modulaires favorisant l'interopérabilité. Les CPS jouent un rôle clé dans la résolution de problématiques sociétales, environnementales et économiques en exploitant des technologies avancées pour améliorer la sécurité, la productivité et l'efficacité [4].

1.1.2 Les systèmes réactifs dans les CPS

Les systèmes réactifs constituent souvent une composante logicielle ou matérielle essentielle des systèmes cyber-physiques (CPS). Les CPS, qui intègrent des dispositifs physiques et des systèmes informatiques, utilisent des systèmes réactifs pour :

— Gérer les stimuli des capteurs.

— Contrôler les actionneurs.

— Assurer une interaction continue avec l'environnement physique.

Par exemple, dans un réseau de voitures autonomes, les systèmes réactifs traitent les données des capteurs pour ajuster la trajectoire ou la vitesse en temps réel.

1.2 Les systèmes réactifs

Dans les années 80, D. Harel et A. Pnueli ont introduit le terme "Systèmes réactifs" [20] Les systèmes réactifs sont des systèmes conçus pour répondre en continu à des événements ou stimuli externes, généralement en temps réel ou quasi-temps réel. Contrairement à d'autres types de systèmes qui fonctionnent de manière autonome ou par traitement par lots, les systèmes réactifs se caractérisent par une interaction permanente avec leur environnement. Leur conception repose sur des mécanismes capables de surveiller, de détecter des changements et de réagir rapidement pour fournir des réponses adaptées.

Parmi les exemples typiques de systèmes réactifs, on trouve les systèmes embarqués dans l'industrie automobile (freinage ABS), les interfaces utilisateur réactives, ainsi

que les réseaux IoT qui collectent et analysent des données en temps réel.

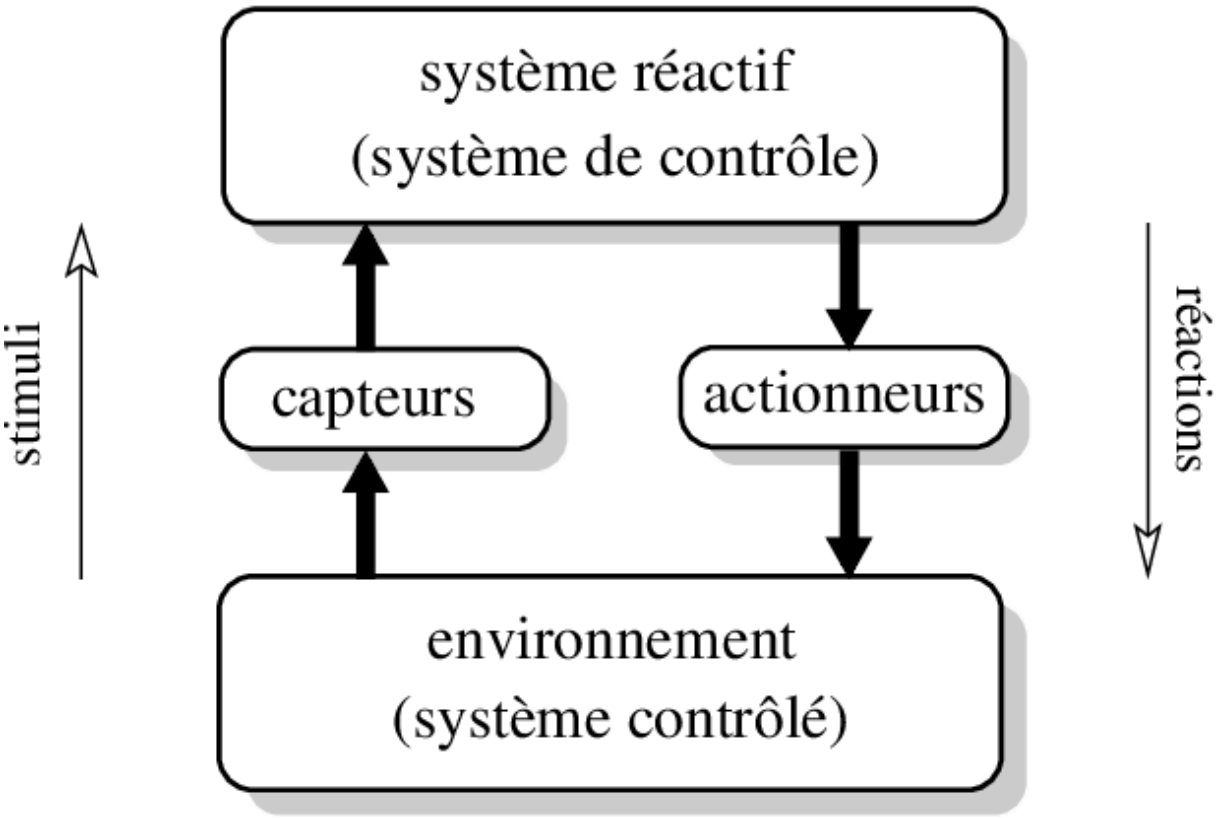

FIGURE 1.1 – Modèle d'un système réactif [25]

1.2.1 Caractéristiques des systèmes réactifs

Les principales caractéristiques des systèmes réactifs sont les suivantes [12] :

a. Réactif

Les systèmes réactifs garantissent une réponse rapide aux événements, minimisant les délais de traitement pour offrir une interaction fluide avec l'utilisateur ou d'autres systèmes. Cette réactivité est essentielle pour maintenir une qualité de service élevée et détecter rapidement les problèmes. Par exemple, dans un système embarqué de freinage ABS, une réponse rapide est critique pour éviter des accidents.

b. Résilient

Les systèmes réactifs restent fonctionnels même en cas de panne grâce à des mécanismes d'isolation, de réplication et de détection précoce des erreurs. La résilience est obtenue en gérant les défaillances à un niveau localisé, permettant ainsi à une partie

du système de continuer à fonctionner malgré les problèmes rencontrés ailleurs. Par exemple, un système IoT dans une maison intelligente peut continuer à fonctionner même si un capteur ou un réseau tombe en panne.

c. Élastique

Les systèmes réactifs s'adaptent dynamiquement aux variations de charge en ajustant les ressources disponibles. Cela permet de maintenir des performances optimales sous une charge croissante tout en réduisant les coûts d'exploitation lorsque la demande diminue. Par exemple, une application de streaming vidéo élastique augmente le nombre de serveurs actifs lorsque le nombre d'utilisateurs connectés augmente.

d. Basés sur les messages

La communication dans les systèmes réactifs est souvent asynchrone, ce qui garantit un couplage lâche et une gestion efficace des erreurs. L'échange de messages entre les composants permet de minimiser les interdépendances, facilitant ainsi la maintenance, l'évolutivité et la tolérance aux pannes. Par exemple, un système de messagerie instantanée, comme WhatsApp, traite les messages de manière asynchrone pour assurer une fluidité dans les interactions même sous forte charge.

1.2.2 Contraintes sur les systèmes réactifs

— **Architecture distribuée** : Les systèmes réactifs sont souvent implémentés dans des architectures distribuées, notamment pour des raisons de rapidité ou de contraintes physiques de distribution (réseaux de capteurs, etc.).

— **Concurrence logique** : Les systèmes peuvent être modélisés à l'aide de processus concurrents, ce qui permet de gérer plusieurs opérations simultanées.

— **Temps réel** : Les systèmes réactifs sont souvent des systèmes temps réel, où non seulement les résultats des calculs sont importants, mais aussi le temps nécessaire pour les obtenir. La précision temporelle est cruciale, car ces systèmes sont soumis à des contraintes de temps strictes, comme des délais de réponse limités et des exigences de disponibilité. Tout résultat ne respectant pas ces contraintes est considéré comme invalide.

— **Hétérogénéité** : Les composants des systèmes réactifs sont souvent réalisés avec des technologies variées, qu'elles soient logicielles ou matérielles. Cette diversité

nécessite une gestion soignée des interactions entre ces différents composants pour assurer la cohésion et le bon fonctionnement du système global.

— **Criticité** : Dans des domaines tels que le transport (aérien ou terrestre), la médecine, ou d'autres secteurs sensibles, les systèmes réactifs sont dits critiques. Un dysfonctionnement peut entraîner de graves conséquences, y compris des risques pour la vie humaine. Ces systèmes nécessitent des garanties élevées en termes de sécurité et de sûreté de fonctionnement, avec une attention particulière portée à la gestion des risques.

1.3 Comparaison avec d'autres types de systèmes

Comme le résume le tableau 1.1, les systèmes transformationnels se caractérisent par une interaction limitée avec l'environnement, se produisant uniquement au début et à la fin du traitement, lequel est généralement séquentiel. Les systèmes interactifs, quant à eux, entretiennent une communication continue avec leur environnement, selon leur propre rythme, ce qui implique un traitement dynamique des informations. Les systèmes temps réel vont plus loin en intégrant des contraintes temporelles strictes : ils doivent réagir dans des délais bien définis, leur traitement pouvant être déterministe ou plus souple selon les exigences. Enfin, les systèmes réactifs se distinguent par une interaction continue étroitement dépendante des stimuli provenant de l'environnement ; leur fonctionnement est typiquement événementiel, c'est-à-dire déclenché par des événements externes qu'ils doivent traiter immédiatement.

Une comparaison des principaux types de systèmes est présentée dans le tableau ci-dessous [46] :

Type de système	Interaction avec l'environnement	Mode de traitement
Transformationnel	Interaction au début et à la fin	Séquentiel
Interactif	Continue, avec vitesse propre	Dynamique
Temps réel	Continue, contraintes temporelles strictes	Déterministe ou souple
Réactif	Continue, piloté par les stimuli de l'environnement	Événementiel

Table 1.1 – Comparaison des systèmes transformationnels, interactifs, temps réel et réactifs

1.4 Exemples de systèmes réactifs

1.4.1 Systèmes embarqués

Les systèmes embarqués dans l'industrie automobile, tels que les régulateurs de vitesse adaptatifs, les systèmes de freinage antiblocage (ABS) et les contrôles de traction, sont des exemples typiques de systèmes réactifs. Ces systèmes intègrent des capteurs, des actionneurs et des calculateurs pour surveiller en temps réel les conditions du véhicule et de la route. Par exemple, l'ABS ajuste la pression de freinage pour éviter le blocage des roues, améliorant ainsi la stabilité et la sécurité du véhicule . De même, les régulateurs de vitesse adaptatifs ajustent automatiquement la vitesse du véhicule en fonction du trafic environnant, contribuant à une conduite plus sûre et plus confortable. Comme le montre la figure 1.2, un modèle typique d'un système embarqué dans l'automobile comprend des composants tels que des capteurs, des actionneurs et des unités de contrôle électronique, tous interconnectés pour assurer une gestion efficace et réactive des diverses fonctions du véhicule.

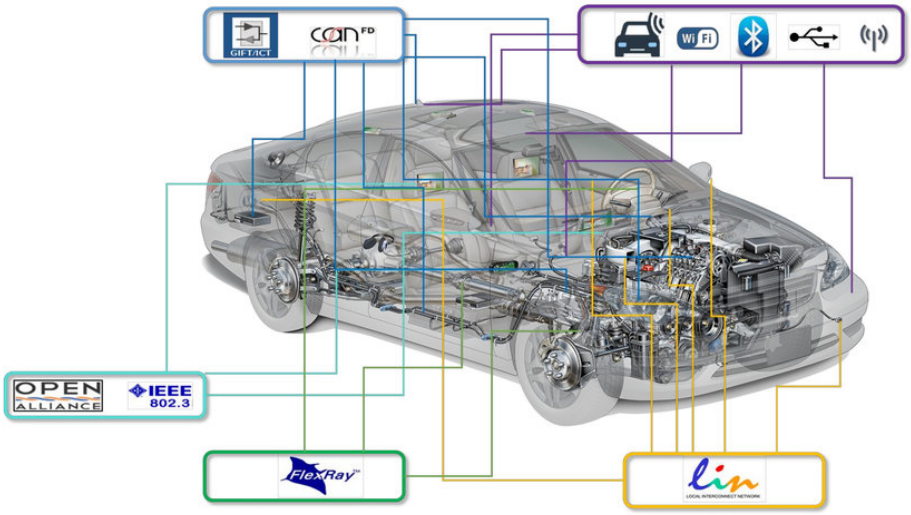

Figure 1.2 – Modèle d'un système embarque[21]

1.4.2 Réseaux IoT

Dans les réseaux de capteurs IoT, les systèmes réactifs collectent, analysent et réagissent aux données en temps réel. Par exemple, dans une maison connectée, des capteurs intelligents surveillent la consommation d'énergie des appareils électroménagers, du chauffage et de l'éclairage. En cas de surconsommation ou d'anomalie, le système peut ajuster automatiquement les paramètres pour optimiser la consommation énergétique, réduisant ainsi les factures d'énergie et l'empreinte carbone de la

maison . Cette approche proactive est également utilisée dans les bâtiments commerciaux et industriels pour améliorer l'efficacité énergétique et la durabilité. Comme le montre la figure 1.3, les réseaux IoT permettent une gestion intelligente et réactive des ressources, contribuant ainsi à des environnements plus durables et économes en énergie.

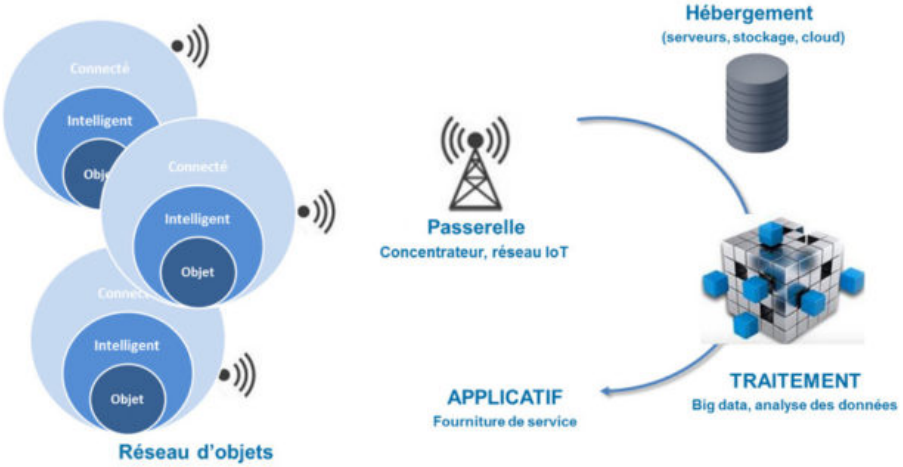

FIGURE 1.3 – Réseaux IoT

1.4.3 Applications basées sur les événements

Les interfaces utilisateur interactives, telles que les jeux vidéo ou les applications bancaires, reposent sur des systèmes réactifs pour offrir une expérience utilisateur fluide et sans latence perceptible. Dans les jeux vidéo, par exemple, les actions du joueur, comme les mouvements ou les commandes, sont traitées instantanément pour mettre à jour l'affichage et la logique du jeu en temps réel. Cela nécessite une conception d'interface utilisateur cohérente et réactive, où les éléments visuels et interactifs s'ajustent dynamiquement en fonction des actions de l'utilisateur, créant ainsi une expérience immersive et engageante . Comme le montre la figure 1.4, les interfaces utilisateur interactives reposent sur des systèmes réactifs pour offrir une expérience utilisateur fluide et sans latence perceptible.

Figure 1.4 – Applications basées sur les événements

1.5 Domaines d'application des systèmes réactifs

Les systèmes réactifs sont omniprésents dans les technologies modernes, offrant des solutions dynamiques et adaptées aux environnements complexes. Leur capacité à traiter des événements en temps réel les rend essentiels dans divers secteurs. Voici un aperçu approfondi de leurs applications :

1.5.1 Industrie automobile

Les véhicules modernes intègrent des systèmes réactifs pour améliorer la sécurité, le confort et l'efficacité énergétique. Des technologies telles que les régulateurs de vitesse adaptatifs, les systèmes de freinage antiblocage (ABS) et les systèmes d'aide à la conduite (ADAS) utilisent des capteurs et des calculateurs pour réagir instantanément aux conditions de la route.

Par exemple, l'ABS ajuste la pression de freinage pour éviter le blocage des roues, tandis que les régulateurs de vitesse adaptatifs modifient la vitesse du véhicule en fonction du trafic environnant. Ces systèmes contribuent à une conduite plus sûre et plus confortable.

1.5.2 Systèmes de contrôle industriel

Dans les chaînes de production automatisées, les systèmes réactifs assurent la synchronisation des machines, la détection des anomalies et l'optimisation des performances en temps réel. Les automates programmables industriels (API) et les systèmes SCADA (Supervisory Control and Data Acquisition) sont des exemples typiques.

Ces systèmes permettent une surveillance continue des processus industriels, une gestion efficace des ressources et une réduction des temps d'arrêt, améliorant ainsi la productivité et la sécurité.

1.5.3 Réseaux de télécommunications

Les infrastructures de communication, notamment les réseaux 5G, utilisent des systèmes réactifs pour gérer le routage des données, la gestion des interférences et l'adaptation dynamique des bandes passantes afin d'assurer une qualité de service optimale.

L'intégration de robots logiciels dotés d'IA permet d'automatiser le traitement des commandes de services, réduisant ainsi les délais et les erreurs, et améliorant l'efficacité opérationnelle des réseaux de télécommunications.

1.5.4 Internet des objets (IoT)

Les maisons intelligentes, les villes connectées et la gestion de l'énergie reposent sur des systèmes réactifs pour collecter, analyser et réagir aux données des capteurs en temps réel. Cela permet d'optimiser la consommation énergétique et d'améliorer le confort des utilisateurs.

Les systèmes IoT peuvent réagir instantanément aux nouvelles informations sans intervention humaine, automatisant des tâches répétitives et améliorant l'efficacité des processus.

1.5.5 Systèmes financiers et bancaires

Les systèmes réactifs sont également cruciaux dans le secteur financier pour le traitement des transactions en temps réel, la détection des fraudes bancaires et la gestion des flux monétaires.

L'utilisation de l'intelligence artificielle permet une surveillance perpétuelle des flux transactionnels, détectant avec précision les anomalies et permettant une intervention proactive face aux menaces potentielles, renforçant ainsi la sécurité des systèmes financiers.

1.5.6 Applications médicales et santé

Les dispositifs médicaux tels que les moniteurs de surveillance des patients, les pompes à perfusion intelligentes et les systèmes d'imagerie médicale nécessitent une réactivité instantanée pour assurer un suivi en temps réel et éviter les situations critiques.

Les systèmes d'information cliniques et les applications mobiles destinées aux professionnels de la santé permettent une gestion efficace des données des patients, améliorant ainsi la qualité des soins et l'engagement des patients.

1.5.7 Jeux vidéo et applications interactives

Les jeux en ligne, les systèmes de streaming et les plateformes interactives utilisent des architectures réactives pour réduire la latence, synchroniser les interactions et gérer dynamiquement la charge des serveurs.

L'intelligence artificielle dans les jeux vidéo permet de créer des comportements réactifs et adaptatifs chez les personnages non-joueurs, offrant ainsi une expérience immersive et engageante pour les joueurs.

En raison de leur capacité d'adaptation et de traitement instantané des événements, les systèmes réactifs sont devenus un élément fondamental dans la conception de solutions intelligentes et performantes dans divers secteurs technologiques et industriels.

1.6 Défis de la conception des systèmes réactifs

1.6.1 Complexité croissante

— **Multiplicité des composants et hétérogénéité** : Les systèmes réactifs intègrent capteurs, actionneurs, interfaces utilisateur et services distants, ce qui complique la coordination et l'interopérabilité.

— **Concurrence et comportement asynchrone** : La gestion de flux d'événements concurrents et de processus asynchrones nécessite des modèles robustes (acteurs, automates I/O) et des outils formels pour éviter conditions de course et assurer la justesse.

— **Évolution et maintenance** : Les exigences évoluent en cours de vie du système. L'architecture doit rester modulaire et extensible pour faciliter l'intégration de nouvelles fonctionnalités.

1.6.2 Contraintes de performance

— **Temps réel et latence** : Les systèmes critiques (contrôle industriel, jeux en ligne) requièrent des réponses dans des délais stricts pour éviter des défaillances graves.

— **Débit et montée en charge** : Certains services doivent traiter des milliers de requêtes par seconde. L'architecture doit évoluer horizontalement tout en préservant cohérence et réactivité.

— **Optimisation des ressources** : Répartition intelligente de la charge, équilibrage dynamique et mise en cache sont indispensables pour maximiser le rendement.

1.6.3 Problématiques de sécurité et sûreté

— **Tolérance aux pannes** : Stratégies de redondance, basculement automatique et récupération après incident garantissent la continuité de service.

— **Sécurité** : Chiffrement, authentification et contrôle d'accès doivent protéger le système des attaques (injection, DDoS, violation de données) sans impacter négativement la performance.

— **Intégrité des données** : Protocoles robustes et gestion des erreurs assurent des informations fiables pour les prises de décision réactives.

1.6.4 Consommation énergétique et durabilité

— **Contrainte énergétique** : Sur les dispositifs IoT ou embarqués, l'optimisation logicielle (veille adaptative, consolidation des messages) minimise la consommation.

— **Empreinte carbone et Green IT** : À l'échelle des centres de données, l'efficacité énergétique (PUE) et l'usage de solutions moins gourmandes réduisent les émissions de CO_2.

— **Conception pour la longévité** : Favoriser l'évolutivité logicielle et la modularité pour prolonger la durée de vie des équipements et réduire les déchets électroniques.

1.7 Fiabilité des systèmes réactifs

La fiabilité constitue un enjeu central dans la conception des systèmes réactifs, particulièrement dans des contextes critiques comme les CPS. En effet, ces systèmes opèrent souvent dans des environnements dynamiques et imprévisibles (réseaux de transport, dispositifs médicaux, etc.), où toute défaillance peut engendrer des conséquences graves, tant sur le plan économique que humain [29].

Plusieurs défis spécifiques compliquent la garantie de fiabilité. D'une part, les contraintes temporelles strictes (temps réel) limitent la possibilité de recourir à des mécanismes de vérification traditionnels. D'autre part, l'hétérogénéité des composants (matériels, logiciels, protocoles) et leur distribution géographique augmentent les risques d'erreurs d'intégration ou de communication. Enfin, la nature événementielle de ces systèmes, où des stimuli externes multiples peuvent survenir de manière concurrente, exige une gestion rigoureuse des états et des transitions pour éviter des comportements indésirables.

À cela s'ajoute la nécessité d'assurer une disponibilité quasi continue, même en cas de perturbation ou de défaillance partielle. La fiabilité ne se limite donc pas à la prévention des pannes, mais inclut également la capacité à détecter, isoler et corriger les erreurs sans perturber l'ensemble du système [3]. Cette exigence est d'autant plus critique dans les systèmes réactifs embarqués dans des infrastructures vitales, où le moindre dysfonctionnement peut avoir des répercussions immédiates.

1.7.1 Approches pour garantir la fiabilité des systèmes réactifs

Face aux exigences croissantes de fiabilité et de sûreté des systèmes réactifs, la conception moderne s'appuie sur un ensemble de méthodologies rigoureuses. Plusieurs approches complémentaires permettent d'atteindre ces objectifs critiques, notamment :

— **Tolérance aux pannes** : Cette technique consiste à doter le système de mécanismes de redondance, qu'ils soient matériels (composants en double) ou logiciels (processus ou services répliqués). En cas de défaillance, le système peut basculer automatiquement sur une ressource saine. Des méthodes comme le rollback (revenir à un état antérieur connu) ou la réplication d'états assurent une continuité de service sans perte de données critiques.

— **Vérification formelle** : Elle repose sur l'utilisation de modèles mathématiques rigoureux pour analyser le comportement du système. Des outils comme les automates temporisés, les réseaux de Petri ou les logiques temporelles permettent de

prouver formellement des propriétés essentielles telles que l'absence de blocage, la vivacité ou la sécurité. Cette approche réduit considérablement les risques d'erreurs non détectées en phase de test classique.

— **Architectures résilientes** : La résilience est assurée par une organisation modulaire du système, où chaque composant peut fonctionner de manière isolée. En cas de défaillance locale, le système global continue de fonctionner avec des performances dégradées mais acceptables. Cette approche favorise également la maintenance et l'évolution du système sans perturber son fonctionnement global.

— **Surveillance proactive** : Cette approche repose sur l'analyse en temps réel des données internes du système pour anticiper les comportements anormaux. Des techniques comme les observateurs d'état ou les algorithmes d'apprentissage automatique permettent de détecter des signes précurseurs de panne. Cela permet d'agir avant que les défaillances ne se produisent, augmentant ainsi la disponibilité du système.

L'intégration de ces méthodes dès les premières phases de conception permet d'anticiper les risques potentiels et de construire des systèmes capables de réagir de manière fiable, même dans des conditions extrêmes.

1.8 Conclusion

Ce chapitre a abordé les systèmes réactifs et leur importance cruciale dans les systèmes cyber-physiques (CPS). Après avoir examiné les caractéristiques, contraintes et interactions avec l'environnement des systèmes réactifs, l'étude a permis de mettre en évidence les exigences techniques nécessaires pour assurer des performances fiables dans les environnements critiques.
L'analyse a souligné l'importance d'une approche rigoureuse dans la modélisation et la validation des composants des CPS, afin de concevoir des modèles robustes et évolutifs répondant aux contraintes spécifiques. Le chapitre suivant portera sur l'étude des modèles formels dédiés aux systèmes réactifs afin de pouvoir proposer des solutions pour améliorer la fiabilité et l'efficacité opérationnelle de ces systèmes.

Chapitre 2

État de l'art sur la modélisation des Systèmes réactifs

Introduction

Parmi les approches utilisées, les modèles dits synchrones occupent une place importante.Ils reposent sur l'hypothèse d'un temps logique global, où les composants réagissent de manière simultanée à chaque "tick" d'horloge. Les I/O Automata, par exemple, permettent de décrire formellement les comportements d'un système en distinguant les actions d'entrée, de sortie et internes. Ce modèle facilite la composition modulaire et la vérification formelle des systèmes distribués, mais il reste limité en termes de scalabilité lorsqu'il s'agit de représenter des interactions complexes ou faiblement synchronisées. Les Réseaux de Kahn (Kahn Process Networks), quant à eux, modélisent des systèmes à travers des processus communiquant via des canaux FIFO. Ils garantissent un comportement déterministe sous certaines hypothèses, ce qui simplifie l'analyse, mais leur incapacité à exprimer des mécanismes de synchronisation explicite constitue une limite dans certains scénarios temps réel.

À l'opposé, les modèles asynchrones, plus proches de la réalité d'exécution de nombreux systèmes embarqués, permettent aux composants de progresser à leur propre rythme, sans synchronisation globale. Cela leur confère une flexibilité intéressante, notamment pour la modélisation des architectures distribuées ou tolérantes aux pannes. Toutefois, cette souplesse introduit des défis importants pour la vérification, car le nombre de comportements possibles croît rapidement avec l'asynchronisme, rendant les analyses formelles plus coûteuses.

Ainsi, chaque modèle présente des compromis entre expressivité, facilité de vérifica-

tion et adéquation avec les contraintes du système cible. De nombreuses recherches visent aujourd'hui à combiner les avantages de ces approches, en proposant des modèles hybrides ou des cadres méthodologiques intégrant à la fois des composants synchrones et asynchrones. Dans ce chapitre, nous explorerons ces différentes approches, en mettant en évidence leurs principes, leurs applications typiques et les limites qu'elles posent dans le contexte des systèmes critiques et temps réel.

2.1 Modélisation des systèmes réactifs

La modélisation des systèmes réactifs est cruciale pour leur conception, leur validation et leur vérification. Ces systèmes, qui interagissent constamment avec leur environnement, doivent satisfaire à des exigences rigoureuses en matière de sécurité, de fiabilité et de contraintes temporelles. Une modélisation approfondie permet de prévoir les comportements potentiels, de détecter d'éventuelles erreurs dès les premières étapes du développement et d'optimiser les performances de manière plus efficace.

Deux approches principales existent pour la modélisation des systèmes réactifs :

— **L'approche asynchrone**, où les composants du système fonctionnent de manière autonome, sans synchronisation stricte. Cette flexibilité facilite la gestion des environnements distribués et des architectures complexes. Toutefois, cette méthode introduit un certain degré de non-déterminisme, rendant la vérification et la validation des comportements du système plus difficiles.

— **L'approche synchrone**, qui repose sur l'idée que les composants du système réagissent de façon coordonnée selon un temps logique. Cette méthode favorise une meilleure prévisibilité et simplifie l'analyse formelle des comportements. En raison de ses avantages en matière de vérification et de certification, elle est souvent privilégiée pour les systèmes critiques, notamment dans les secteurs de l'aéronautique, de l'automobile et des systèmes embarqués.

Dans ce document, nous nous focalisons sur l'approche synchrone, qui offre des garanties renforcées en matière de sécurité et de validation pour les systèmes critiques.

2.1.1 Approche asynchrone

L'approche asynchrone constitue un concept clé dans le domaine de l'informatique et des systèmes distribués. À la différence des systèmes synchrones, où toutes les opérations sont régulées par une horloge commune, les systèmes asynchrones permettent aux différents composants d'un système d'opérer de manière autonome. Cette ap-

proche a été introduite pour la première fois par Martin (1989) dans le cadre de la conception d'un microprocesseur asynchrone [38].

a. Principes

Un modèle asynchrone se caractérise par plusieurs principes fondamentaux :

— **Indépendance temporelle** : Les composants peuvent échanger des messages sans nécessiter une synchronisation stricte, permettant ainsi une exécution décentralisée.

— **Communication par message** : Les interactions entre composants s'effectuent principalement via des messages, généralement de manière non bloquante. L'expéditeur d'un message n'est pas contraint d'attendre une réponse pour poursuivre son exécution.

— **Flexibilité et résilience** : Cette architecture offre une meilleure adaptation aux variations de latence et aux changements dynamiques des conditions du système. Elle présente également une tolérance accrue aux pannes, une défaillance locale n'entraînant pas nécessairement l'interruption de l'ensemble du système.

Comme le montre la figure 2.1, la communication asynchrone à commande uniforme permet aux producteurs d'événements de publier des messages sur un bus d'événements, auxquels plusieurs consommateurs peuvent s'abonner et réagir indépendamment, favorisant ainsi un découplage efficace et une scalabilité accrue.

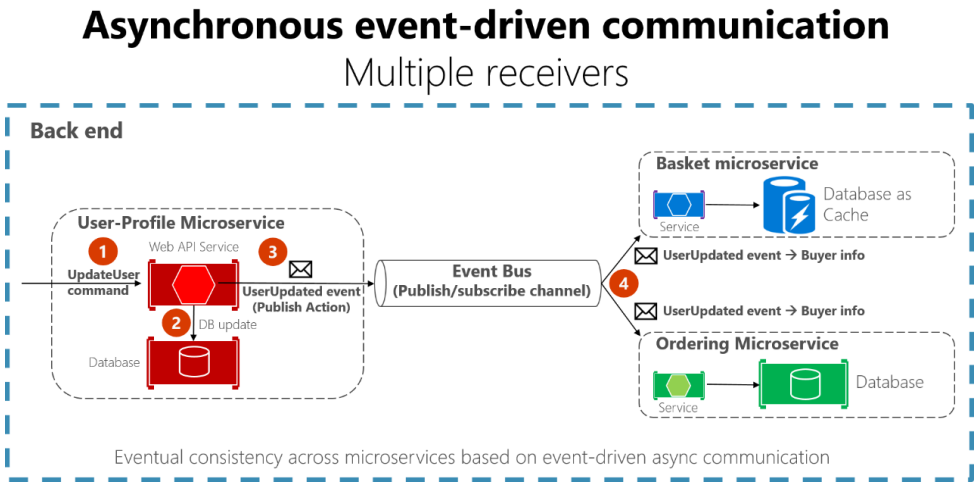

FIGURE 2.1 – Schéma illustrant la communication asynchrone à commande uniforme [39]

b. Applications des modèles asynchrones

Les modèles asynchrones trouvent une application étendue dans de nombreux secteurs. Dans le cadre des **systèmes distribués**, ils permettent une communication entre plusieurs nœuds sans exiger une synchronisation rigoureuse, ce qui est particulièrement pertinent dans les architectures **microservices** et **les services web**. Par exemple, les échanges HTTP entre un client et un serveur illustrent un modèle asynchrone, où le serveur peut traiter plusieurs requêtes de clients simultanément sans provoquer de blocage.

Dans le domaine du **développement web**, l'asynchronisme joue un rôle crucial pour optimiser l'expérience utilisateur. Les applications ont la capacité de charger des données en arrière-plan, tout en maintenant l'interface graphique opérationnelle. Des technologies telles que **JavaScript** (avec les fonctionnalités **async/await**), **WebSocket**, ainsi que des systèmes de gestion de messages comme **RabbitMQ** et **Kafka**, contribuent à la création d'architectures asynchrones efficaces.

Comme le montre la figure suivante, une architecture typique de microservices illustrant des communications asynchrones utilise des files de messages ou des bus d'événements pour permettre aux services de communiquer de manière découplée et résiliente.

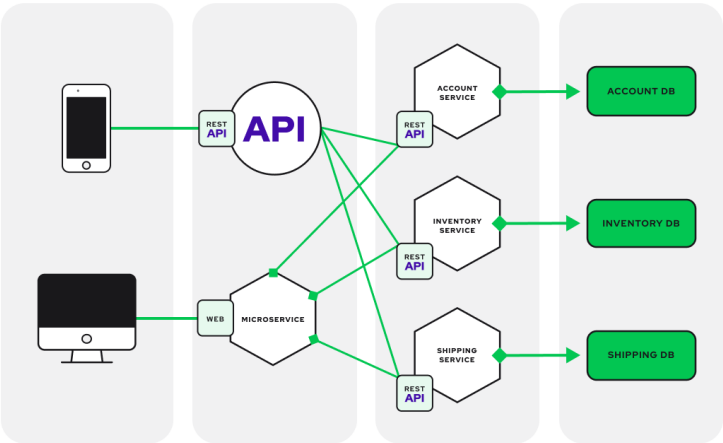

Figure 2.2 – Architecture typique des microservices illustrant des communications asynchrones [39]

c. Avantages et limitations

Avantages

Les modèles asynchrones offrent plusieurs bénéfices notables :

— **Amélioration des performances** : En permettant le traitement simultané de plusieurs tâches, ils maximisent l'utilisation des ressources et augmentent le débit global du système.

— **Scalabilité** : L'extension d'un système asynchrone se fait plus aisément sans perturber son fonctionnement, ce qui est crucial pour les grandes infrastructures cloud et les applications à forte parallélisation.

— **Réactivité** : Grâce à une gestion des tâches non bloquante, les applications asynchrones peuvent fournir une expérience utilisateur plus fluide et dynamique.

Limitations

Néanmoins, ces modèles sont limités par rapport aux points suivants :

— **Complexité de la programmation** : La gestion des interactions entre les composants asynchrones peut s'avérer plus complexe à appréhender, notamment en raison des problèmes de courses de données et de synchronisation des événements.

— **Difficulté de débogage** : Les erreurs dans un système asynchrone peuvent être ardues à détecter et à reproduire, car elles proviennent souvent d'interactions non linéaires entre divers processus.

2.1.2 Approche synchrone

L'approche synchrone est fondée sur l'idée d'un temps discret, ce qui implique que l'exécution du système est segmentée en moments successifs, chaque réaction étant considérée comme instantanée. Cette hypothèse facilite la modélisation et la vérification des systèmes réactifs en garantissant une synchronisation précise des événements. Comme l'indique N. Halbwachs (1993) [18] dans son livre "Synchronous programming of reactive systems", cette méthode assure un comportement déterministe et prévisible des systèmes.

Les modèles synchrones se distinguent par une exécution où toutes les actions sont coordonnées selon un même rythme temporel. À la différence des modèles asynchrones, où les composants interagissent de manière indépendante, les modèles synchrones requièrent une coordination stricte des interactions. Cela contribue à une fiabilité accrue et simplifie l'analyse formelle du système.

Plusieurs formalismes illustrent cette approche :

— **Les réseaux de Petri synchrones** : Ces outils permettent de modéliser des systèmes réactifs en imposant des contraintes temporelles strictes sur les transitions.

— **Les automates synchrones** : Ces dispositifs sont utilisés pour représenter des comportements séquentiels et réactifs, facilitant ainsi la spécification et la vérification des systèmes critiques.

— **Les langages synchrones** : Parmi eux, on trouve Esterel, Lustre et Signal, chacun étant conçu pour garantir une exécution déterministe et une vérification formelle efficace.

L'approche synchrone se distingue des modèles asynchrones par son caractère déterministe et sa capacité à assurer un comportement prévisible, ce qui est crucial pour les systèmes embarqués et critiques. Toutefois, cette rigidité peut également poser des défis lors de la modélisation d'environnements où les interactions sont imprévisibles.

2.1.3 Histoire et Origine des Modèles Synchrones

Les premiers modèles synchrones apparus dans les années 1980 s'appuient sur l'hypothèse du temps logique, où les événements sont traités à des moments discrets et synchronisés. Cette méthode garantit une exécution déterministe, un aspect crucial pour les systèmes critiques. C'est dans ce contexte que plusieurs langages synchrones ont été créés pour répondre aux besoins de modélisation et de programmation des systèmes embarqués :

Esterel : Conçu par Gérard Berry et al.[7], Esterel est un langage impératif destiné à la programmation réactive. Il repose sur une exécution synchrone des instructions et permet une génération efficace de code matériel et logiciel. Son utilisation est particulièrement répandue dans les secteurs de l'aérospatiale et de l'automobile.

Lustre : Proposé par Caspi et Halbwachs[19], Lustre est un langage déclaratif fondé sur les flux de données. Il est principalement utilisé pour la spécification et la vérification des systèmes embarqués, ayant une influence significative sur le développement de SCADE, un outil industriel dans le domaine aéronautique.

Signal : Introduit par Le Guernic et al.[28], Signal est un langage synchrone axé sur la gestion des communications et des systèmes temps réel distribués. Il se distingue par

sa capacité à gérer plusieurs horloges synchronisées, facilitant ainsi la modélisation des interactions complexes entre les composants matériels et logiciels.

Au cours des décennies 1990 et 2000, les bases théoriques des modèles synchrones ont été solidifiées. La formalisation des sémantiques opérationnelles et dénotationnelles a accru leur précision mathématique, tandis que les I/O Automata, présentés par Lynch et Tuttle, ont élargi leur capacité d'expression. Simultanément, l'émergence d'outils de vérification formelle, tels que le model checking, ainsi que l'intégration des concepts de composition modulaire, ont favorisé leur adoption croissante dans divers domaines industriels et académiques.

FIGURE 2.3 – Histoire des modèles synchrones[11]

2.1.4 Comparaison entre les systèmes synchrones et asynchrones

Les systèmes synchrones et asynchrones se distinguent principalement dans leur gestion temporelle, comme le montre la figure 2.4 illustre la différence entre les modes de transmission synchrone, où l'envoi et la réception de données sont coordonnés par une horloge commune, et asynchrone, où les données sont transmises sans signal d'horloge partagé, chaque élément fonctionnant à son propre rythme. Mais aussi la coordination des tâches et la communication entre composants. Le choix entre une approche ou une autre dépend largement du contexte de l'application, des exigences en termes de performance, de fiabilité et de complexité. Le tableau 2.1 ci-dessous présente une synthèse des principales différences entre ces deux types de systèmes.

Aspect	Système Synchrone	Système Asynchrone
Coordination temporelle	Basée sur une horloge commune	Indépendante, sans horloge commune
Exécution des tâches	Séquentielle, chaque tâche attend la fin de la précédente	Parallèle, les tâches s'exécutent simultanément
Avantages	Prévisibilité, simplicité de mise en œuvre	Flexibilité, meilleure utilisation des ressources
Inconvénients	Risque de blocage en cas de retard	Complexité accrue, gestion des erreurs plus difficile
Applications typiques	Applications financières, industrielles nécessitant une coordination stricte	Interfaces utilisateur, applications web, systèmes de messagerie

TABLE 2.1 – Comparaison entre les systèmes synchrones et asynchrones

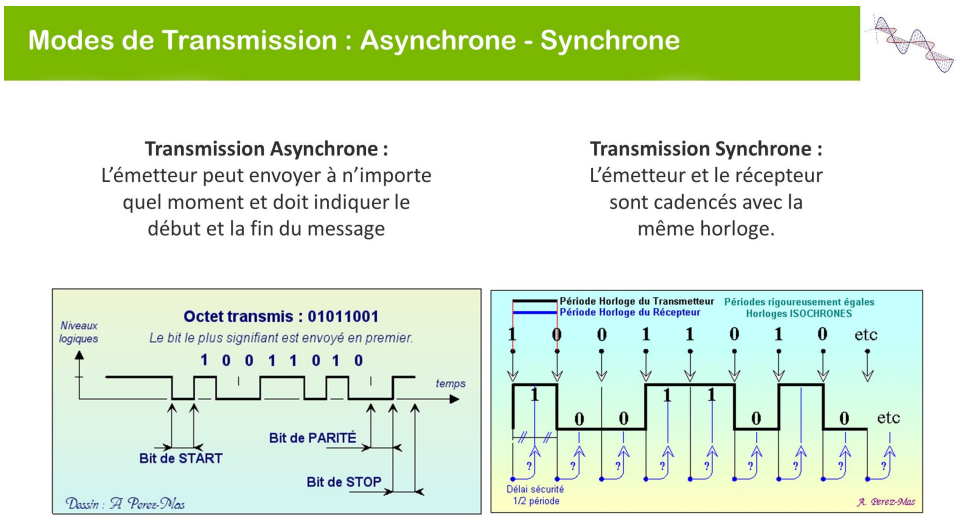

FIGURE 2.4 – Modes de Transmission : Asynchrone -Synchrone[44]

2.2 Modèles de conception synchrones

Les modèles synchrones constituent des formalismes couramment employés dans la modélisation des systèmes réactifs et distribués. Ils garantissent un comportement déterministe, ce qui est crucial pour assurer la fiabilité et la sécurité des systèmes critiques. Ces modèles s'appuient sur une synchronisation rigoureuse entre les divers composants du système, facilitant ainsi la validation et la vérification formelle des propriétés du système.

Nous allons analyser en profondeur trois exemples représentatifs de modèles synchrones : les Automates I/O [35], les Réseaux de Kahn (KPN) [24] et le SDL (Speci-

fication and Description Language) [23]. Chacun de ces modèles joue un rôle essentiel dans plusieurs domaines, allant de la conception de protocoles de communication à l'optimisation des systèmes embarqués et cyber-physiques.

2.3 Automate

Avant d'introduire le modèle utilisé dans le cadre de notre travail, nous commençons par introduire la notion d'automate : Un **automate** est un dispositif ou une machine capable d'exécuter automatiquement une séquence d'actions ou d'opérations prédéterminées sans intervention humaine. Il peut s'agir d'un objet programmé avec une mémoire, qui reproduit souvent des mouvements ou comportements, parfois imitant ceux d'un être vivant. Historiquement, les automates mécaniques utilisaient des systèmes comme leviers, poulies, engrenages, et se sont développés avec l'horlogerie. Le terme peut aussi désigner familièrement une personne agissant de manière mécanique ou inconsciente.

Dans le domaine de l'informatique théorique, un automate est une *machine abstraite* qui traite des informations discrètes (comme des symboles ou des mots sur un alphabet fini). Il possède un nombre fini d'états internes et évolue selon des règles déterministes ou non, en fonction des entrées et de son état courant. Les automates peuvent avoir des mémoires auxiliaires (par exemple : pile, bande infinie) selon leur type, comme les automates finis, les automates à pile, ou les machines de Turing. Ils sont utilisés pour modéliser la computation, l'analyse syntaxique et la théorie des langages formels [37].

2.4 Les automates d'entrée/sortie (I/O Automata)

Dans le cadre de notre travail, nous avons choisi d'utiliser les automates d'entrée/sortie (I/O Automata) car ils permettent de modéliser de manière claire et structurée les interactions entre composants dans un système réactif. Ce formalisme est particulièrement adapté à notre étude, car il distingue explicitement les actions d'entrée, de sortie et internes, ce qui facilite la modélisation des échanges entre les différents éléments du système. Les **automates d'entrée/sortie** (*Input/Output Automata*, ou *I/O automata*) ont été introduits par Lynch et Tuttle [35] afin de modéliser les systèmes à événements discrets. Basés sur les machines à états, les I/O automata ont été largement utilisés dans la spécification et l'analyse de nombreux types de systèmes.

Le modèle original, introduit en 1987, concerne les *I/O automata asynchrones* [36], appliqués à la modélisation d'algorithmes d'allocation de ressources dans les réseaux, à

différents niveaux d'abstraction. Ce modèle a depuis été largement adopté dans l'étude des systèmes distribués.

Une extension significative, proposée par Lynch et Vaandrager [45], a introduit les *I/O automata temporisés*, qui permettent de prendre en compte la dimension temporelle dans les transitions. Ces modèles ont été appliqués à l'analyse de performances, la détection de pannes, le consensus, et la synchronisation d'horloges.

Les *I/O automata hybrides* [34] représentent une autre évolution importante. Ils permettent de modéliser des systèmes présentant à la fois des comportements discrets et continus. Ils ont été appliqués à des études variées, allant de modèles jouets à des systèmes complexes tels que le contrôle d'hélicoptères.

En 1995, Segala a introduit les *I/O automata probabilistes* [42], qui intègrent des choix probabilistes et non déterministes, notamment pour les protocoles de sécurité ou les systèmes distribués aléatoires.

Enfin, les *I/O automata dynamiques*, proposés par Attie [2], permettent la création et la suppression dynamique de processus, offrant une modélisation plus souple de systèmes adaptatifs.

2.4.1 Propriétés des I/O automata

Les automates d'entrée/sortie présentent plusieurs caractéristiques fondamentales :

— **Déterminisme** : Chaque action engendre un état distinct, garantissant ainsi la prévisibilité du système.

— **Modularité** : Il est envisageable d'associer plusieurs automates afin de modéliser des systèmes complexes de manière structurée.

— **Vérification formelle** : Il permet une analyse approfondie des comportements potentiels du système et assure qu'il respecte les exigences de sécurité.

2.4.2 Description des I/O automata

Un *automate d'entrée/sortie* (I/O automaton) est défini par un ensemble d'actions, qui sont : les **entrées**, les **sorties** et les **actions internes**. Ces actions constituent l'interface entre l'automate et son environnement.

Formellement, un ensemble d'actions $acts(S)$, où S est une signature d'actions, est une partition en trois sous-ensembles :

— $in(S)$: les actions d'entrée,

— $out(S)$: les actions de sortie,

— $int(S)$: les actions internes.

L'ensemble des actions **externes**, noté $ext(S)$, est défini par :

$$ext(S) = in(S) \cup out(S)$$

Une signature d'actions externes est une signature ne contenant pas d'actions internes.

Les actions **contrôlées localement**, notées $local(S)$, sont définies par :

$$local(S) = out(S) \cup int(S)$$

Ces actions sont sous le contrôle local d'un automate ayant S comme signature d'actions.

D'après la définition classique de Lynch et Tuttle [35], un *I/O automaton* A est constitué des éléments suivants :

— une signature d'actions $sig(A)$,

— un ensemble d'états $states(A)$,

— un ensemble non vide d'états initiaux $start(A) \subseteq states(A)$,

— une relation de transition $steps(A) \subseteq states(A) \times acts(A) \times states(A)$.

Nous nous appuyons ici sur la définition proposée par Arnold [1], qui présente une version simplifiée et opérationnelle des I/O automata, bien adaptée à notre cadre de modélisation.

Définition 1 (I/O automaton) *Un I/O automaton est un quadruplet $A = \langle S, s_0, \Sigma, \rightarrow \rangle$, où :*

— *S est l'ensemble des états,*

— *$s_0 \in S$ est l'état initial,*

— *Σ est l'ensemble des actions d'entrée et de sortie,*

— $\rightarrow \subseteq S \times \Sigma \times S$ *est la relation de transition.*

Chaque transition $(s, l, s') \in \rightarrow$ *est notée* $s \xrightarrow{l} s'$. *L'étiquette* l *représente une action d'entrée ou de sortie exécutée lors de la transition. Si une action* l *n'est pas exécutable depuis un état* s, *on note* $s \nrightarrow$.

2.5 Input/Output Automata et ses Variations

Les Automates d'Entrée/Sortie (I/O Automata) ont été introduits par Nancy Lynch et Mark Tuttle en 1989 [35] pour la modélisation de systèmes concurrents et distribués. Ces automates permettent de formaliser les interactions entre des composants communicants en définissant des états et des transitions, ce qui les rend adaptés à la spécification de systèmes critiques. Les I/O Automata constituent des machines à états finis où, à chaque transition, le système reçoit des données d'entrée provenant de l'environnement et produit des données de sortie représentant les nouvelles valeurs calculées à partir des entrées. Bien qu'initialement introduits pour un cadre applicatif spécifique, leur usage a été généralisé à diverses catégories de systèmes réactifs.

a. Timed I/O Automata

Les Timed I/O Automata [26] étendent le modèle de base en incorporant des contraintes temporelles. Cela permet de modéliser des systèmes en temps réel où le comportement dépend non seulement de la séquence des événements, mais aussi du moment où ils se produisent.

b. Probabilistic I/O Automata

Les Probabilistic I/O Automata [33] intègrent des transitions probabilistes pour modéliser l'incertitude dans le comportement des systèmes. Ils sont utilisés notamment pour l'analyse de la fiabilité des systèmes distribués confrontés à des défaillances aléatoires.

c. Hybrid I/O Automata

Les Hybrid I/O Automata [31] combinent des dynamiques discrètes et continues. Ce modèle est conçu pour représenter les systèmes cyber-physiques où les phénomènes physiques (dynamiques continues) interagissent avec des composants logiciels (dynamiques discrètes).

d. Dynamic I/O Automata

Les Dynamic I/O Automata ont été introduits par Qadeer et al. [41] pour modéliser des systèmes dont la structure peut évoluer au cours du temps (ajout ou suppression de composants, modification des connexions). Cette approche est particulièrement pertinente pour les systèmes dynamiques et adaptatifs tels que les CPS évolutifs.

2.5.1 Caractéristiques et usages des I/O Automata

Parmi les **caractéristiques essentielles des I/O Automata**, on peut citer :

— Le **déterminisme**, qui assure une exécution prévisible et qui est une caractéristique primordiale des systèmes synchrones

— La **modularité**, qui permet de combiner plusieurs automates pour modéliser des systèmes complexes,

— La **vérification formelle**, qui facilite une analyse rigoureuse des comportements possibles et leur conformité aux exigences.

Les I/O Automata sont utilisés dans **divers domaines** de l'ingénierie des systèmes, notamment pour :

— La **vérification de protocoles de communication**,

— La **modélisation de systèmes embarqués critiques** (par exemple, dans l'aéronautique et l'automobile),

— L'**analyse des systèmes cyber-physiques**, qui intègrent des composants matériels et logiciels,

— La **validation d'algorithmes distribués**, où la synchronisation et la cohérence sont primordiales.

2.5.2 Réseaux de Kahn (KPN)

Les Réseaux de Kahn, introduits par G. Kahn en 1974 [24], constituent un modèle déterministe pour la modélisation des systèmes réactifs fondés sur l'échange de flux de données à travers des canaux FIFO. Ils sont élaborés autour de processus autonomes qui interagissent sans nécessiter d'horloge globale, assurant ainsi un comportement modulaire, extensible et répétable, indépendamment de l'ordre d'exécution. Ce modèle est à l'origine de nombreux langages synchrones tels que Lustre ou Signal [30],

et se retrouve dans des domaines variés comme le traitement du signal, les systèmes embarqués ou la simulation parallèle.

Parmi leurs principaux avantages :

— Un déterminisme fort,

— Une absence de blocage global,

— Une grande modularité.

Néanmoins, leur mise en œuvre pratique est restreinte par l'hypothèse de canaux à mémoire infinie et la complexité de l'ordonnancement optimal. Malgré ces limitations, les KPN demeurent une référence essentielle dans la modélisation des systèmes concurrents.

Figure 2.5 – Réseaux de Kahn (KPN)[27]

2.5.3 SDL (Specification and Description Language)

Le SDL [23] est un langage formel qui a été normalisé par l'UIT-T, et qui a été conçu pour spécifier, simuler et valider des systèmes réactifs ainsi que des systèmes en temps réel. Il s'appuie sur des machines à états finis communicantes, alliant une notation graphique intuitive à une sémantique formelle rigoureuse. Grâce à sa structure modulaire et hiérarchique, le SDL simplifie la conception de systèmes complexes, en particulier dans les secteurs de la télécommunication, de l'aéronautique et de l'automobile. Il permet de représenter les communications asynchrones entre processus, de modéliser des comportements concurrents, et de générer automatiquement du code exécutable à partir des modèles.

Parmi ses atouts :

— Support de la concurrence et des communications événementielles,

— Modélisation formelle adaptée aux systèmes embarqués critiques,

— Génération automatique de code à partir de modèles validés.

Cependant, il présente des limites dues à une courbe d'apprentissage élevée et à la complexité de la gestion des modèles de grande taille sans outils spécialisés.

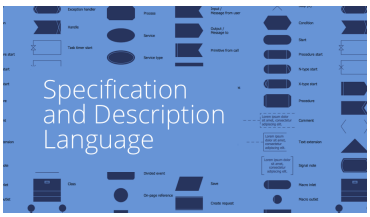

FIGURE 2.6 – SDL (Specification and Description Language)

2.5.4 Réseaux de Petri synchrones

Les *Réseaux de Petri* constituent un formalisme à la fois graphique et mathématique, développé par Carl Adam Petri dans les années 1960 [40], dans le but de modéliser des systèmes de traitement d'informations parallèles. Ils se composent de places, qui symbolisent des états ou des ressources, de transitions, qui illustrent des événements ou des actions, et d'arcs reliant les places aux transitions. Leur principal atout réside dans leur aptitude à représenter de manière intuitive la concurrence, la synchronisation et la causalité au sein des systèmes réactifs.

Dans le cadre des modèles synchrones, on recourt à des variantes connues sous le nom de *Réseaux de Petri synchrones* ou *Réseaux de Petri temporels synchrones*, où l'exécution des transitions est régie par une horloge globale ou logique [9]. Ce synchronisme exige que les composants du système progressent simultanément à chaque cycle, contrairement aux Réseaux de Petri traditionnels, souvent employés dans des contextes asynchrones. Cette contrainte temporelle garantit un comportement déterministe du système, ce qui est essentiel pour les systèmes critiques, tels que ceux utilisés dans l'automobile, l'aéronautique ou les processus industriels.

Un des avantages majeurs des Réseaux de Petri synchrones est leur potentiel d'analyse formelle. Il est possible d'appliquer des outils mathématiques pour vérifier des propriétés essentielles telles que l'accessibilité, la vivacité, l'absence de blocage et la sécurité. Ces caractéristiques revêtent une importance particulière dans les systèmes embarqués et les systèmes temps réel, où des erreurs d'exécution peuvent entraîner des conséquences graves.

En matière de modularité, les Réseaux de Petri peuvent être organisés de manière hiérarchique ou enrichis par des modules [5], ce qui facilite la modélisation de systèmes complexes tout en préservant une structure claire. Ils sont fréquemment associés à d'autres formalismes, tels que les langages synchrones comme *Esterel*, ou servent de

backend d'analyse pour des outils industriels, comme dans certaines configurations de *SCADE*.

Cependant, ce formalisme présente également des limites. La représentation graphique peut rapidement devenir complexe pour des systèmes de grande envergure. Par ailleurs, la modélisation d'un temps réel précis, incluant des délais et des échéances, peut nécessiter l'emploi de réseaux de Petri étendus ou de réseaux temporisés, ce qui complique l'analyse. Malgré cela, leur capacité d'expression, leur rigueur mathématique et leur compatibilité avec les outils de vérification automatique en font un modèle essentiel dans l'analyse des systèmes réactifs synchrones.

2.6 Conception par Composition des systèmes réactifs

Dans la conception des systèmes réactifs et cyber-physiques, la modularité est une nécessité incontournable pour gérer la complexité croissante des interactions entre les éléments logiciels et matériels. Les systèmes contemporains se composent de sous-ensembles fonctionnels interconnectés, souvent élaborés par des équipes distinctes et dans des environnements variés. Il est donc impératif d'adopter une approche de conception qui permette d'assembler des modules indépendants afin de créer un système global cohérent, vérifiable et résilient [6].

La conception par composition repose sur la description séparée des comportements des composants ainsi que de leurs interactions, puis de les combiner selon des règles formelles. Cette approche assure la garantie de propriétés globales (telles que la sûreté, l'absence de blocage ou la cohérence temporelle) à partir des caractéristiques locales des sous-systèmes. Elle encourage également la réutilisation, la portabilité, la testabilité et la durabilité des composants au fil du temps [43].

Pour atteindre ces objectifs, plusieurs plateformes de modélisation ont été mises au point. Parmi celles-ci, *Ptolemy II*, *SCADE* et *BIP* se distinguent par leur capacité à intégrer les principes de composition dans des contextes concrets, adaptés à la modélisation, à la simulation, à la vérification, et parfois à la génération automatique de code. Chaque outil offre une perspective unique sur la composition, en fonction de son domaine d'application et de ses bases théoriques.

2.6.1 Ptolemy II

Ptolemy II est un logiciel développé par l'Université de Berkeley, aux États-Unis [16]. Il a pour objectif de modéliser, simuler et analyser des systèmes embarqués hétérogènes. Ce projet s'inscrit dans une démarche de recherche académique, visant à fournir un

cadre flexible pour explorer divers modèles de calcul au sein d'un même environnement. Le concept fondamental de Ptolemy repose sur l'intégration de composants aux comportements variés – tels que synchrones, asynchrones, à temps discret ou continu – au sein d'une architecture logicielle unique. Cette capacité à gérer l'hétérogénéité des composants répond directement à la complexité croissante des systèmes modernes, en particulier dans les domaines des systèmes cyber-physiques, de l'IoT et de la robotique.

Forces et limites

Ptolémée II représente une plateforme robuste pour la modélisation et la simulation de systèmes complexes grâce à son intégration de divers modèles de calcul (synchrone, asynchrone, événementiel, continu) au sein d'un environnement unifié. Il se distingue par sa capacité à modéliser efficacement des systèmes cyber-physiques et des architectures embarquées hétérogènes. Sa conception modulaire, basée sur les concepts d'acteurs et de directeurs, facilite la structuration, la hiérarchisation et la réutilisation des composants. De plus, son interface graphique intuitive et son caractère open source en font un outil prisé dans les milieux académiques pour le prototypage et la recherche. Cependant, Ptolémée II présente certaines limitations qui entravent son adoption dans l'industrie. Il ne propose ni générateur de code certifié ni vérification formelle avancée, ce qui le rend inadapté aux environnements critiques. Son interface peut s'avérer complexe pour les débutants, et l'absence de support officiel ainsi que de conformité aux normes industrielles strictes limite son utilisation dans les projets nécessitant une certification rigoureuse.

2.6.2 SCADE

SCADE (Safety Critical Application Development Environment) est un environnement de développement destiné à la modélisation, la simulation, la vérification et la génération de code pour les systèmes embarqués critiques, notamment dans les domaines de l'aéronautique, de l'automobile et de l'énergie. Il repose sur le langage synchrone Lustre, ce qui permet de garantir un comportement déterministe et prévisible des applications [17].

SCADE permet la description précise de la logique des composants, tout en intégrant des mécanismes de vérification formelle tels que la génération d'obligations de preuve ou la vérification d'invariants. Il est certifié selon différentes normes de sûreté comme DO-178C (aéronautique), ISO 26262 (automobile) ou CEI 61508 (industrie), ce qui en fait un outil industriel de référence pour le développement de systèmes critiques [10].

2.6.3 BIP

Le langage BIP (Behavior, Interaction, Priority) est un cadre formel pour la modélisation et la composition de systèmes hétérogènes. Il repose sur une architecture en trois couches distinctes [6] :

— **La couche Comportement**, qui représente les éléments fondamentaux sous la forme d'automates à états finis enrichis de données internes.

— **La couche Interaction**, qui établit les modes de communication entre les composants (synchronisations, messages, etc.).

— **La couche Priorité**, qui permet d'établir un ordre d'exécution pour les interactions concurrentes, afin de maîtriser de manière précise le comportement du système global.

Cette distinction offre une grande flexibilité dans la conception : les composants peuvent être développés de manière autonome, et leurs interactions peuvent être définies par la suite sans nécessiter de modifications de leur logique interne. Ce principe confère à BIP une modularité remarquable, particulièrement appréciée pour l'expérimentation de différentes architectures ou politiques d'exécution à partir d'un même ensemble de composants.

Forces et limites

Le cadre BIP (Comportement, Interaction, Priorité) se caractérise par son architecture en couches qui dissocie clairement le comportement, les interactions et les priorités. Cette organisation favorise la clarté, la modularité et la réutilisation des composants, tout en simplifiant la maintenance et l'expérimentation de diverses configurations architecturales. BIP repose sur une base formelle robuste, ce qui en fait un outil rigoureux pour la modélisation, particulièrement adapté aux systèmes réactifs et distribués. Il propose également des mécanismes de validation automatique pour vérifier des propriétés complexes, un avantage considérable dans le développement de systèmes critiques. Cependant, BIP est encore peu utilisé dans l'industrie, notamment en raison de son manque d'intégration avec les outils standards et de son interface peu intuitive. De plus, sa maîtrise requiert une bonne compréhension des concepts formels, ce qui limite son accessibilité aux non-spécialistes et restreint son utilisation au milieu académique.

2.7 Positionnement de nos travaux

L'analyse des modèles synchrones et des approches formelles associées a permis de mettre en évidence les contributions théoriques essentielles à la modélisation des systèmes réactifs.

Dans une étude fondatrice, Johan Eker et ses collaborateurs [16] ont proposé une modélisation des systèmes hétérogènes en introduisant une architecture basée sur la coordination de modèles de calcul. Leur contribution repose sur une technique de composition modulaire hiérarchique, permettant de décomposer un système en sous-composants exécutés selon des règles de synchronisation précises. Ce cadre a ouvert la voie à des approches comme les SR-modèles, qui introduisent une abstraction par deux paramètres : la mémoire (M) et la latence (L), pour représenter les comportements synchrones tout en maintenant une modularité.

Dans une autre étude importante, Halbwachs et al. [17] ont proposé une approche formelle de modélisation par flot de données synchrones. Leur travail se base sur une exécution déterministe pilotée par un temps logique global, et une composition en série des modules selon un flux unidirectionnel. Cette technique permet de garantir la correction par construction lorsque les composants sont chaînés linéairement. Cependant, elle reste limitée à des connexions séquentielles et ne permet pas de représenter des interactions parallèles ou multiples.

Dans un prolongement théorique, Halbwachs [18] a formalisé la modélisation des systèmes réactifs synchrones à l'aide de logiques temporelles et de modèles de transition. Il y développe une technique rigoureuse de spécification comportementale permettant la vérification formelle de propriétés telles que la sûreté, l'absence de blocage ou la vivacité. Néanmoins, cette approche suppose une description complète du système, souvent sous forme de code, et reste orientée vers une composition séquentielle des modules.

Notre travail s'inscrit dans la continuité de ces recherches, tout en répondant à leurs limites. Nous proposons une extension des SR-modèles en définissant un opérateur de composition parallèle, permettant de modéliser des architectures où plusieurs composants consomment un même signal simultanément. Cette contribution repose sur une technique formelle de synchronisation parallèle des transitions, garantissant un comportement déterministe et vérifiable. Contrairement aux approches précédentes, notre modèle ne nécessite ni l'accès au code source, ni l'usage d'un langage synchrone, rendant la modélisation plus accessible et mieux adaptée à des systèmes partiellement spécifiés.

2.8 Conclusion

Dans ce chapitre, nous avons présenté les principaux modèles synchrones permettant de représenter le comportement des systèmes réactifs, en mettant en avant leur capacité à gérer les interactions, la synchronisation et la composition entre composants. Nous avons également examiné plusieurs environnements de conception basés sur la modularité, tout en soulignant les limites liées à leur complexité et à la nécessité d'une expertise technique. Ces constats nous ont amenés à proposer, dans le chapitre suivant, une approche plus accessible et modulaire, fondée sur des modèles paramétrés par la mémoire et la latence, pour faciliter la modélisation et la vérification des systèmes cyber-physiques.

Chapitre 3

Contribution : Composition parallèle des SR-modèles

Introduction

Dans le chapitre précédent, nous avons effectué une analyse des approches formelles de modélisation et vérification des systèmes réactifs, nous nous intéressons principalement à la modélisation des systèmes réactifs synchrones et nous proposons une extension de l'un des modèles existants.

Dans ce chapitre, nous présentons notre contribution : un nouvel opérateur de composition qui permet de concevoir des SR-modèles(Synchronous reactive models) corrects-par-construction. Pour cela, nous commençons par définir les systèmes réactifs synchrones étudiés, puis nous présentons leur modélisation. Nous introduisons ensuite les I/O Automata et leurs règles de composition. Après avoir identifié les limites des approches actuelles, nous formulons la problématique spécifique que notre proposition vise à résoudre, Enfin, nous présentons notre proposition fondée sur une extension des I/O-automata : les SR-modèles.

3.1 Systèmes réactifs synchrones

Un *système réactif* est un système informatique qui interagit en continu avec son environnement, réagissant à des événements externes dans des délais stricts. Contrairement aux systèmes transformationnels, qui traitent des données de manière séquentielle, les systèmes réactifs doivent répondre instantanément aux stimuli de leur environnement, ce qui les rend essentiels dans des domaines tels que le contrôle industriel,

les systèmes embarqués et les applications critiques [18].

Les *systèmes réactifs synchrones* (SR-systèmes) se distinguent par l'hypothèse d'un temps logique global, où toutes les réactions au sein du système se produisent simultanément à chaque "tick" d'horloge. Cette approche permet de simplifier la conception et la vérification des systèmes en éliminant les non-déterminismes liés à l'ordonnancement des événements [8].

Parmi les paramètres principaux qui permettent de décrire le comportement des SR-systèmes, nous nous intéressons aux paramètres suivants :

— **Latence (L)** : le délai entre l'arrivée des entrées et la production des sorties correspondantes.

— **Mémoire (M)** : la capacité de stockage disponible pour les données internes du système.

Ces systèmes sont souvent modélisés à l'aide de langages de programmation synchrones tels que *Esterel*, *Lustre* et *SIGNAL*, qui offrent des garanties de déterminisme et facilitent la vérification formelle [18].

Les SR-systèmes sont largement utilisés dans des applications nécessitant des garanties de temps réel et de fiabilité, telles que les systèmes de contrôle de vol, les dispositifs médicaux et les systèmes de sécurité industriels. Leur capacité à réagir de manière prévisible et rapide aux événements externes les rend particulièrement adaptés à ces environnements exigeants.

3.2 Modélisation

L'intégration de techniques de vérification formelle dans la conception de systèmes nécessite l'adoption d'un formalisme adapté à l'analyse formelle. Nous présentons ici le modèle choisi pour décrire naturellement le comportement des SR-composants. Ces modèles permettent la vérification de propriétés fonctionnelles et s'inscrivent dans une approche compositionnelle, facilitant leur combinaison pour former des systèmes complexes cohérents et complets.

3.2.1 Abstraction des données

Dans ce travail, nous proposons un formalisme de haut niveau pour modéliser les systèmes synchrones réactifs à un niveau transactionnel. L'objectif est d'abstraire des valeurs réelles des données pour ne considérer que les événements de transfert, ce qui

est particulièrement pertinent dans les systèmes de traitement de signal ou les architectures orientées flux de données, où le traitement ne dépend pas du contenu des données, mais uniquement de leur présence ou absence.

Cette modélisation est également bien adaptée aux systèmes dans lesquels le flot d'entrées pilote les traitements effectués. Ainsi, les composants sont vus comme des entités réactives encapsulées, placées dans un environnement qui peut leur fournir des données en entrée et recevoir celles proposées en sortie. Le composant peut accepter ou refuser des entrées selon son état interne.

Le modèle en boîte noire permet d'analyser un composant uniquement à travers ses ports d'interaction, sans considération de sa structure interne. Cette abstraction facilite considérablement la vérification formelle et la composition de systèmes hétérogènes (voir Figure 3.1).

Dans cette approche, nous considérons également la mémoire interne des composants ainsi que la latence de traitement. Bien que la modélisation soit effectuée selon une perspective abstraite des données, l'état interne du composant, qui représente sa mémoire, a un impact direct sur sa capacité à recevoir de nouvelles entrées ou à générer des sorties. De plus, certaines interactions peuvent exiger un délai entre l'entrée et la sortie, ce qui reflète une latence inhérente au traitement du signal ou à la logique interne du système. Ces éléments sont cruciaux pour représenter fidèlement le comportement temporel des systèmes réactifs synchrones, où les réactions ne sont pas toujours instantanées et dépendent de l'historique des interactions.

Figure 3.1 – Vue en boîte noire d'un composant réactif

3.3 Composition des I/O Automata

Notre étude porte sur la conception des systèmes réactifs par approche compositionnelle. Cette méthodologie consiste à créer un système en assemblant divers composants, chacun étant conçu de façon autonome. La représentation de ces systèmes peut se faire sous forme de graphes, où nous identifions deux catégories distinctes de connexions entre les composants.

a. Connexion en série (bout à bout)

Dans cette configuration les composants sont connectées comme dans un pipeline.

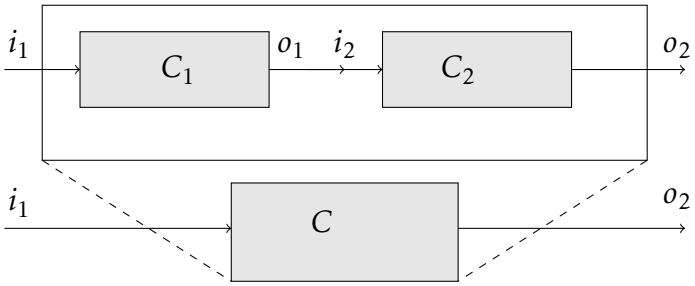

FIGURE 3.2 – Composition en série de deux composants [12]

L'illustration de la figure 3.2 représente une configuration en série dans laquelle le composant C_1 peut accepter des entrées (illustré par : i_1) transmet ses sorties (O_1) comme entrées (i_2) au composant C_2. L'entrée globale i_1 est traitée de manière séquentielle par les deux composants afin de générer une sortie finale $0-2$. Ce schéma représente une architecture de type pipeline, fréquemment employée pour garantir un traitement structuré des données.

b. Connexion en parallèle(côte à côte)

Une connexion en parallèle de deux composants signifie que les entrées des deux composants sont connectées au même point , et que leurs sorties sont également connectées ensemble . Cela crée deux chemins distincts entre l'entrée et la sortie du système, permettant au signal de circuler simultanément à travers les deux composants.

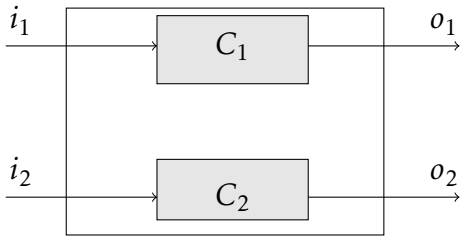

FIGURE 3.3 – Composition en parallèle de deux composants

L'illustration de la figure 3.3 représente une configuration en parallèle dans laquelle un seul composant C est capable de recevoir plusieurs flux d'entrées distincts (illustrés par : i_1 et i_2) de manière simultanée. Chaque entrée est traitée indépendamment au sein du composant, générant respectivement des sorties propres (o_1 et o_2). Cette structure permet un traitement concurrent des données, ce qui peut considérablement améliorer les performances globales du système, notamment dans des contextes où les tâches sont indépendantes ou partiellement découplées. Ce type d'architecture est couramment utilisé dans les systèmes réactifs ou embarqués pour paralléliser les opérations et réduire la latence de traitement.

Dans ce travail, nous voulons adapter la composition des I/O-automata à la connexion en parallèle. L'avantage d'utiliser les I/O Automata (Automates d'Entrée/Sortie) réside dans leur capacité de composition : la possibilité d'assembler des automates de manière modulaire, assurant un comportement harmonieux du système global basé sur celui des sous-composants. Pour cela, nous allons d'abord donner la définition des règles de composition classique des I/O automata (Input/Output Automata). Dans sa forme classique [32], cette composition repose sur deux règles de synchronisation.

Définition 2 (Composition des I/O-automata) *Soient deux I/O automata :*

$$A_1 = \langle S_1, s_{01}, \Sigma, \rightarrow_1 \rangle \quad et \quad A_2 = \langle S_2, s_{02}, \Sigma, \rightarrow_2 \rangle,$$

leur composition produit un automate composite :

$$A = \langle S_1 \times S_2, (s_{01}, s_{02}), \Sigma, \rightarrow \rangle,$$

où la fonction de transition $\rightarrow$ est définie par les deux règles suivantes :

— ***Règle 1 — Synchronisation directe(Transfert)*** *:Une transition peut se réaliser lorsque le premier composant est apte à générer une sortie(o) et que le second est disposé à recevoir cette sortie en entrée (i). Cela permet un transfert effectif de données*

entre les deux composants.

$$(s_1, s_2) \xrightarrow{\alpha/\beta} (s_1', s_2') \quad si \quad s_1 \xrightarrow{\alpha/o} s_1' \quad et \quad s_2 \xrightarrow{i/\beta} s_2'.$$

— ***Règle 2 — Avancement local sans transfert*** *Une transition peut également se produire sans transfert de données, c'est-à-dire lorsque le premier composant n'émet aucune donnée ($\bar{o}$) et que le second n'en consomme aucune($\bar{i}$). Cette règle illustre des évolutions autonomes des composants.*

$$(s_1, s_2) \xrightarrow{\alpha/\beta} (s_1', s_2') \quad si \quad s_1 \xrightarrow{\alpha/\bar{o}} s_1' \quad et \quad s_2 \xrightarrow{\bar{i}/\beta} s_2'.$$

Limites de la composition classique des I/O Automata dans les SR-systèmes :

L'application de la composition classique des I/O-Automata (Définition 2) aux systèmes réactifs synchrones (SR-systèmes) présente plusieurs limitations structurelles. Ces systèmes, qui se caractérisent par un déroulement synchrone des événements, généralement sous forme de cycles d'horloge, nécessitent un comportement déterministe, ordonné et temporellement cohérent entre les composants. Cependant, la composition standard, qui repose sur la synchronisation des actions partagées, révèle rapidement ses insuffisances dans ce contexte.

— **États indésirables :** peuvent survenir, certaines configurations menant à des combinaisons d'états incohérentes en raison d'un manque de synchronisation adéquate entre les automates. [13]

— **Comportements indéterministes :** peuvent se manifester, où plusieurs transitions deviennent possibles simultanément sans critères de sélection clairs, compromettant ainsi la reproductibilité et la fiabilité du système. [13]

— **Désorganisation du flux de données :** est particulièrement critique dans des architectures de type pipeline, où un désalignement entre les rythmes d'émission et de réception de données peut provoquer des blocages ou une perte de données. [13]

Ces limitations ont été minutieusement identifiées et analysées par Chabane et al [13] . Les auteurs mettent en lumière deux phénomènes spécifiques qui illustrent ces dysfonctionnements.

— **Bubbly States (États à bulles)** se manifestent lorsque des transitions inactives sont intégrées dans le flux de données, entravant ainsi le déroulement normal des

événements valides. Ce phénomène crée un vide logique qui bloque le processus de traitement sans justification fonctionnelle valable. [13]

— **Untidy States (États désorganisés) :** quant à eux, apparaissent lorsque l'ordre temporel des événements est perturbé, rendant l'exécution erratique et compliquant l'analyse du comportement par rapport aux contraintes temporelles attendues dans les systèmes réactifs synchrones. [13]

Les auteurs concluent que la composition classique ne convient pas à ces systèmes et qu'il est impératif de réévaluer les mécanismes de composition pour maintenir la cohérence temporelle et fonctionnelle. Ils suggèrent une nouvelle approche de composition qui intègre des contraintes temporelles explicites, garantissant ainsi une progression fluide et déterministe du système. [13]

3.4 Problématique

Comme nous avons précisé dans la section précédente, la composition classique des I/O-automata ne donne pas le résultat attendu. Bien que cette approche soit adaptée aux systèmes distribués, elle ne l'est pas pour les systèmes réactifs synchrones. En effet, quand on combine deux I/O automata avec les règles définies dans la Définition 2, l'automate obtenu ne correspond pas au résultat attendu [13]. En particulier :

— Les contraintes temporelles ne sont pas correctement modélisées.

— Génère des états inconsistants, qui compromettent la sûreté du système.

— Introduit de l'indéterminisme et ainsi introduit de l'incertain dans des systèmes totelment déterministes.

Ces insuffisances rendent cette composition inappropriée à la conception de systèmes synchrones réactifs sûres et corrects par construction. Les auteurs [12] ont proposé un modèle appelé SR-modèle avec un opérateur de composition qui préserve la correction des systèmes synchrones réactifs par composition.

Cependant, cet opérateur ne prend en compte qu'une configuration des composants, qui est la connexion en série des composants, ce qui ne traduit pas toutes les configurations possibles des composants.

Dans notre travail, nous proposons d'étendre les SR-modèles, en proposant un opérateur de composition parallèle, ce qui permettra de représenter une configuration plus complète des SR-systèmes.

3.5 Proposition

Comme le but de ce travail est de concevoir des systèmes réactifs par composition. Notre travail se base sur les SR-modèles [14] qui se basent sur les I/O automata en étant dédiés à la modélisation des systèmes réactifs. Les auteurs ont défini un SR-modèle primitif pour représenter un composant grâce à ses paramètres, ainsi qu'un opérateur de composition adapté aux connexions en série.

De notre côté, nous proposons de nouvelles règles de composition, qui permettent de modéliser des systèmes plus complexes incluant des connexions en parallèle, et ainsi permettre de représenter une classe plus grande de systèmes réactifs.

Dans le cadre de ce travail, nous nous penchons sur la connexion en parallèle de composants recevant des signaux à partir d'une même source, qui correspond à un mode de communication en Multicast où un composant source diffuse les signaux à plusieurs composants qui traitent les signaux reçus en parallèle, indépendamment les uns des autres.

3.5.1 Modèle choisi : SR-modèles

Dans le cadre de ce travail, nous nous basons sur les SR-modèles(Synchronous-Reactive models) [14] une extension des I/O-automata dédiées à la modélisation des systèmes synchrones réactifs.

Les SR-modèles représentent les composants grâce à deux paramètres fondamentaux :

— **M** : qui représente **la mémoire** utilisée pour conserver un historique partiel des états antérieurs nécessaires à la prise de décision.

— **L** : qui correspond à **la latence** maximale, définie comme le nombre de cycles entre la réception d'une entrée et la génération de la sortie associée.

Nous donnons dans ce qui suit la définition formelle des SR-modèles [14]

Définition 3 (SR-modèles) *Pour chaque composans C ayant comme paramètres M et L, Le SR-modèle correspondant A est décrit comme le quadruplet :*

$$A = \langle S, s_0, \Sigma, \rightarrow \rangle$$

où :

— *S désigne l'ensemble des états accessibles,*

— s_0 *l'état initial,*

— Σ *l'ensemble des actions (entrées/sorties),*

— $\rightarrow$ *la relation de transition, paramétrée par* M *et* $L = \{i/o, i/\bar{o}, \bar{i}/o, \bar{i}/\bar{o},\}$.

Les actions de Σ ***sont définies comme suit :***

— i *: présence d'une entrée,*

— $\bar{i}$ *: absence d'entrée,*

— o *: présence d'une sortie,*

— $\bar{o}$ *: absence de sortie.*

Ces modèles assurent que les sorties sont produites dans un délai minimal de L cycles après la réception d'une entrée, tout en prenant en compte un historique des entrées afin de respecter la mémoire M.

3.5.2 Exemple :

a. SR-modèles primitifs ($L = 1, M = 1$) :

Considérons un SR-modèles caractérisé par les paramètres $L = 1$ et $M = 1$, illustré dans la Figure 3.4. Ce composant peut être dans l'un des deux états suivants :

— $\langle\,\rangle$: le composant est vide.

— $\langle 1\rangle$: une donnée est présente dans le composant.

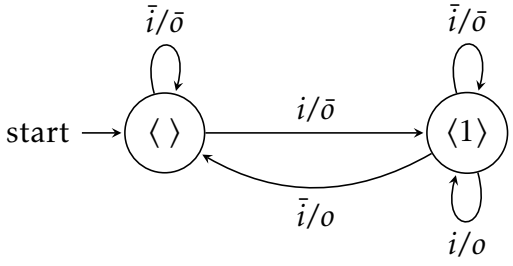

Figure 3.4 – SR-modèle avec $L = 1$, $M = 1$

Description du comportement du SR-modèle : Lorsque le composant est vide, c'est-à-dire en état $\langle\,\rangle$, il peut soit :

— Attendre (transition en boucle avec $\bar{i}/\bar{o}$), ce qui signifie qu'aucune donnée n'est reçue ni transmise,

— Recevoir une donnée (transition $i/\bar{o}$), ce qui le fait passer à l'état $\langle 1 \rangle$ pour indiquer qu'une donnée est maintenant stockée.

À l'état $\langle 1 \rangle$, le composant contient une donnée. Plusieurs scénarios sont possibles :

— Le composant peut attendre sans recevoir ni émettre de donnés ($\bar{i}/\bar{o}$),

— Le composant peut produire une sortie sans nouvelle entrée ($\bar{i}/o$) et revenir à l'état initial ($\langle\rangle$),

— Le composant peut simultanément émettre une donnée et en recevoir une autre (i/o).

Ainsi, le comportement du composant est entièrement décrit par ces deux états et cinq transitions, représentant les comportements d'entrée/sortie du composant.

b. SR-modèles primitifs ($L = 2, M = 2$)

Considérons un SR-modèles caractérisé par les paramètres $L = 2$ et $M = 2$, illustré dans la Figure 3.5.

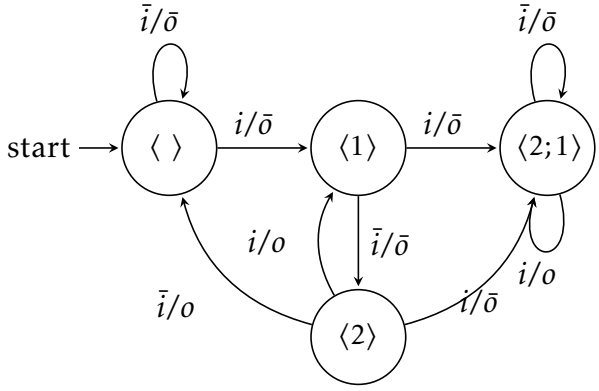

Figure 3.5 – SR-modèle avec $L = 2$, $M = 2$

Description du comportement du SR-modèle : Lorsque le composant est vide, c'est-à-dire en état $\langle\ \rangle$, il peut soit :

— Attendre (transition en boucle avec $\bar{i}/\bar{o}$), ce qui signifie qu'aucune donnée n'est reçue ni transmise,

— Recevoir une donnée ($i/\bar{o}$), ce qui le fait passer à l'état $\langle 1 \rangle$, indiquant qu'une donnée est désormais stockée.

À l'état $\langle 1 \rangle$, le composant contient une donnée. Plusieurs scénarios sont possibles :

— Il peut attendre sans recevoir de nouvelle entrée ($\bar{i}/\bar{o}$), le menant à l'état $\langle 2 \rangle$, qui représente le calcul du temps qui passe,

— Il peut recevoir une deuxième donnée ($i/\bar{o}$), menant à l'état $\langle 2;1 \rangle$.

L'état $\langle 2;1 \rangle$ représente un état où deux entrées successives ont été reçues (mémoire pleine) :

— Il peut continuer à recevoir des entrées en produisant des sorties simultanément (i/o) et rester sur le même état,

— Il peut produire une sortie sans entrée ($\bar{i}/o$) et passer à l'état $\langle 2 \rangle$,

— Attendre sans rien recevoir ni transmettre ($\bar{i}/\bar{o}$), tout en restant dans le même état.

Depuis l'état $\langle 2 \rangle$, trois comportements sont possibles :

— Retourner à l'état initial en produisant une sortie sans nouvelle entrée ($\bar{i}/o$),

— Produire une sortie tout en recevant une nouvelle entrée (i/o), ce qui mène à l'état $\langle 1 \rangle$,

— Recevoir une entrée sans produire de sortie ($i/\bar{o}$), menant à l'état $\langle 2;1 \rangle$.

Ainsi, l'automate modélise un composant réactif capable de mémoriser jusqu'à deux données en entrée et d'adapter son comportement en fonction de sa mémoire et latence.

3.5.3 Composition en série des SR-modèles

Dans le cadre de la modélisation des systèmes réactifs synchrones (SR-modèles), la composition en série constitue une stratégie essentielle pour assembler plusieurs composants en un système global tout en préservant leurs propriétés fondamentales [14]. Cette méthode offre plusieurs avantages significatifs pour la fiabilité des systèmes intégrés. Premièrement, elle assure la conservation du déterminisme : les transitions sont établies de manière à prévenir tout comportement non déterministe, ce qui est essentiel dans les systèmes en temps réel. Deuxièmement, elle respecte rigoureusement les contraintes de latence, garantissant que les délais de production des sorties demeurent

conformes aux attentes, indépendamment du chemin emprunté dans le système intégré. Enfin, tous les états accessibles du système résultant sont valides, ce qui signifie que la composition évite la création d'états incohérents ou irréalistes dus à des interactions mal maîtrisées entre les composants. Ces garanties fournies par l'opérateur de composition en série sont essentielles pour le développement de systèmes corrects par conception.

Formellement : Deux SR-modèles représentant deux composants connectés en série ayant respectivement M_1, L_1 et M_2, L_2 génèrent un SR-modèle ayant comme paramètres $M = M_1 + M_2$ et $L = L_1 + L_2$.

a. Exemple de composition Séquentielle avec ($L1 = 1$, $M1 = 1$) et ($L2 = 1$, $M2 = 1$) :

Lorsqu'on connecte deux composants en série, l'un alimentant directement l'autre, leur interaction devient strictement synchronisée : la sortie du premier constitue l'entrée du second. Dans cette configuration, les transitions sont coordonnées via les actions intermédiaires partagées, permettant ainsi la construction d'un automate global combinant les paramètres de mémoire et latence comme suit :

— La **latence cumulée** est de $L = L_1 + L_2 = 2$, car chaque composant a une latence de 1.

— La **mémoire totale** est également $M = M_1 + M_2 = 2$.

États :

— $\langle\ \rangle$: aucun des deux éléments ne contient de donnée.

— $\langle 1\,;0\rangle$: la donnée est en cours de propagation, encore dans le premier composant.

— $\langle\ 1\,;0\,;1\rangle$: Les deux composants contiennent chacun une donnée (en file).

— $\langle 0\,;1\ \rangle$: la donnée a progressé vers le second composant, le premier étant vide.

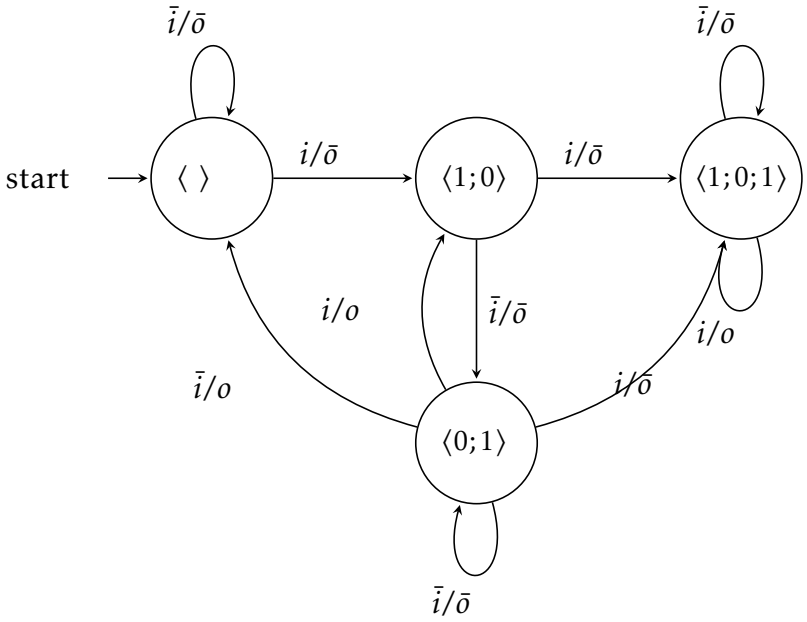

FIGURE 3.6 – SR-modèle composite avec $L = 2$, $M = 2$

Description du comportement de la composition en série de SR-modèles : La figure 3.6 représente le comportement de la composition de deux SR-modèles ayant chacun une mémoire = 1 et une latence = 1 comme montré précédemment (voir figure 3.4), les états composites se comportent comme suit :

— **État initial** $\langle\ \rangle$: cet état combine les deux états initiaux des SR-modèles primitifs.

  — Le système composite peut rester inactif si aucune donnée n'arrive ($\bar{i}/\bar{o}$).

  — Une donnée peut arriver vers le premier composant, provoquant la transition vers $\langle 1;0\rangle$ ($i/\bar{o}$).

— **État $\langle 1;0\rangle$** :

  — Une entrée supplémentaire peut être reçue ($i/\bar{o}$), si le premier composant fait sortir une donnée et la transfert vers le deuxième composant menant à $\langle 1;0;1\rangle$,

  — Ou bien la donnée progresse d'un niveau (interne) sans nouvelle entrée ($\bar{i}/\bar{o}$), vers $\langle 0;1\rangle$.

— **État $\langle 1;0;1\rangle$** :

  — Le système reste silencieux sans interaction ($\bar{i}/\bar{o}$) et reste sur le même état.

— Il peut aussi simultanément produire une sortie depuis le deuxième composant et accepter une nouvelle donnée dans le premier composant (i/o), conservant l'état inchangé.

— Il peut produire une sortie sans nouvelle entrée et passer à l'état $\langle 0;1\rangle$

— **État $\langle 0;1\rangle$** :

Plusieurs comportements sont possibles :

— Une entrée et une sortie simultanées ramènent à $\langle 1;0\rangle$ (i/o),

— Absence d'interactions, on reste stable ($\bar{i}/\bar{o}$),

— Une sortie sans entrée ramène à l'état initial $\langle\ \rangle$ ($\bar{i}/o$),

— Ou une entrée sans sortie conduit à $\langle 1;0;1\rangle$ ($i/\bar{o}$).

Ce **comportement synchronisé** démontre une avancée organisée des données à travers deux composants consécutifs. Chaque transition représente une modification d'état fondée sur l'interaction entre l'entrée, la propagation interne et la sortie. Le modèle valide les propriétés cruciales telles que le déterminisme, le respect du délai maximal, la réceptivité, et la capacité à rester inactif en l'absence d'événements.

Cependant, cette communication ne représente qu'une partie des configurations possibles des composants réactifs synchrones. Dans la suite, nous passons à notre contribution, qui est consacrée à la composition parallèle des composants réactifs synchrones.

3.6 Contribution : Composition parallèle des SR-modèles

3.6.1 Motivation

Dans le cadre de cette contribution, nous proposons un nouvel opérateur de composition parallèle, cet opérateur vise à modéliser un composant complexe par composition qui englobe plusieurs composants connectés en parallèle.

Nous nous inéressons à la configuration présentée dans la figure 4.11. Dans cette architecture, les composants fonctionnent de manière parallèle et reçoivent simulta-

nément les données (ou signaux) provenant d'une source unique, ce qui constitue un modèle de communication **Multicast**.

3.6.2 Principe de la composition en parallèle

Prenons deux composants C_1 ayant les paramètres M_1, L_1 et C_2 ayant les paramètres M_2, L_2 connectés en parallèle. Leur composition produit un composant C ayant les paramètres M, L qui reçoit deux signaux à partir d'une source C_0 (Figure 4.11) illustrés par i_1 et i_2 qui seront traités respectivement par C_1 et C_2.

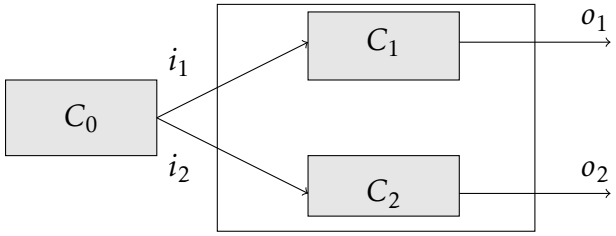

FIGURE 3.7 – Connexion en parallèle type Multicast

Hypothèse sur les paramètres M et L

Soit C_1 et C_2 deux SR-composants, définis respectivement par (M_1, L_1) et (M_2, L_2). Le modèle global C est caractérisé par les paramètres (M, L), définis comme suit :

— **Mémoire globale :**

$$M = \min(M_1, M_2)$$

Comme mentionné précédemment, la connexion que nous traitons (Figure 4.11) est une connexion Multicast, vu que C_1 et C_2 reçoivent les signaux à partir de la même source, le composant ayant la plus petite mémoire définit le comportement du composant composite (les cas où la composition peut accepter les données).

— **Latence globale :**

$$L = \begin{cases} \max(L_1, L_2) & \text{si } L_1 > 1 \text{ ou } L_2 > 1 \\ 1 & \text{si } L_1 = 1 \text{ ou } L_2 = 1 \end{cases}$$

Lorsque deux composants traitent simultanément des données reçues au même moment, la latence totale du système composite correspond à celle du composant le plus lent.

3.6.3 Définition de la composition parallèle des SR-modèles

Dans le but de modéliser le comportement de la connexion parallèle de deux SR-composants qui reçoivent des données d'une même source (Multicast) nous allons utiliser un nouvel opérateur de composition $C_1 \parallel_p C_2$ qui va générer un SR-modèle C dont :

— Les états sont des paires d'états (s_1, s_2) où s_1 représente un état de C_1 et s_2 un état de C_2 ;

— Les transitions résultent de la synchronisation des transitions de C_1 et C_2, en fonction de leurs actions respectives.

Cette composition simule une diffusion de signaux entre les composants (Multicast), où un composant initie la communication avec C_1 et C_2 en parallèle.

3.6.4 Définition formelle de la composition parallèle

La définition formelle des règles de composition respecte la sémantique et les propriétés des SR-modèles tel que proposé dans [14]. En effet, le résultat obtenu par composition parallèle (SR-modèle composite) de C_1 avec M_1, L_1 et C_2 avec M_2, L_2, a le même comportement que le SR-modèle primitif C ayant $M = min(M_1, M_2)$ et $L = max(L_1, L2)$.

Avant d'introduire la définition formelle de la composition, nous allons introduire des notations utiles pour la suite :

— Composant inactif (vide) :

$empty(s)$: Signifie que le composant est dans l'état initial (vide).

— Mémoire actuelle du composant :

$m(s)$: le nombre de données présentes dans le composant à l'état s

— Latence actuelle du composant :

$max_L(s)$: Le temps d'attente de la données la plus ancienne

Définition 4 (Composition parallèle des SR-modèles) *Prenons deux SR-modèles A_1 et A_2. La composition parallèle est définie par l'opérateur $\parallel_p$ est un quadruplet : $A_1 \parallel_p A_2 = \langle S_1 \times S2, (s_{01}, s_{02}), \Sigma, \rightarrow \rangle$ tel que la fonction de transition $\rightarrow$ est définie par les règles suivantes :*

Si

$empty(s_1)$ *et* $not(empty(s_2))$, ***ou***

$empty(s_2)$ *et* $not(empty(s_1))$, ***ou***

$m(s'_1) \neq m(s'_2)$ ***ou***

$max_L(s_1) \neq max_L(s_2)$ *et* $(max_L(s_1) < L_1 \vee max_L(s_2) < L_2)$,

Alors

$(s_1, s_2) \nrightarrow$ *: Signifie qu'aucune transition composée n'est générée.*

Sinon

1. $(s_1, s_2) \xrightarrow{i/o} (s'_1, s'_2) \Rightarrow s_1 \xrightarrow{i/o} s'_1, s_2 \xrightarrow{i/o} s'_2$
2. $(s_1, s_2) \xrightarrow{i/\bar{o}} (s'_1, s'_2) \Rightarrow s_1 \xrightarrow{i/\bar{o}} s'_1, s_2 \xrightarrow{i/\bar{o}} s'_2$
3. $(s_1, s_2) \xrightarrow{\bar{i}/o} (s'_1, s'_2) \Rightarrow s_1 \xrightarrow{\bar{i}/o} s'_1, s_2 \xrightarrow{\bar{i}/o} s'_2$
4. $(s_1, s_2) \xrightarrow{\bar{i}/\bar{o}} (s'_1, s'_2) \Rightarrow s_1 \xrightarrow{\bar{i}/\bar{o}} s'_1, s_2 \xrightarrow{\bar{i}/\bar{o}} s'_2$

Dans la composition parallèle, une transition entre états composites $(s_1, s_2) \xrightarrow{\alpha/\beta} (s'_1, s'_2)$ n'est générée que si les deux composants effectuent simultanément une transition locale compatible avec l'étiquette (α, β), où α et β correspondent aux actions des interfaces externes (α : entrées et β : sorties) des deux SR-composants.

Pour garantir la cohérence et la synchronisation des actions, celles-ci sont soumises à des contraintes spécifiques. La composition n'est pas réalisée lorsque l'un des composants est vide tandis que l'autre contient des données, ou lorsque le nombre de données diffère entre les deux composants. Par ailleurs, les délais de traitement des deux composants doivent être compatibles,puisque les données ont été reçus simultanément.

Quand ces contraintes sont respectées, la composition peut se faire pour générer les quatre types de transitions : (i/o), $(i/\bar{o})$, $(\bar{i}/o)$, et $(\bar{i}/\bar{o})$.

Ces règles assurent une composition rigoureuse et garantissent le respect de la sémantique déterministe des SR-modèles.

3.7 Exemple illustratif de composition parallèle

Nous illustrons dans la figure 3.8 le résultat de la composition parallèle de deux composants $C_1(M_1 = 1, L_1 = 1)$ et C_2 $(M_2 = 1, L_2 = 1)$ en appliquant la définition 4.

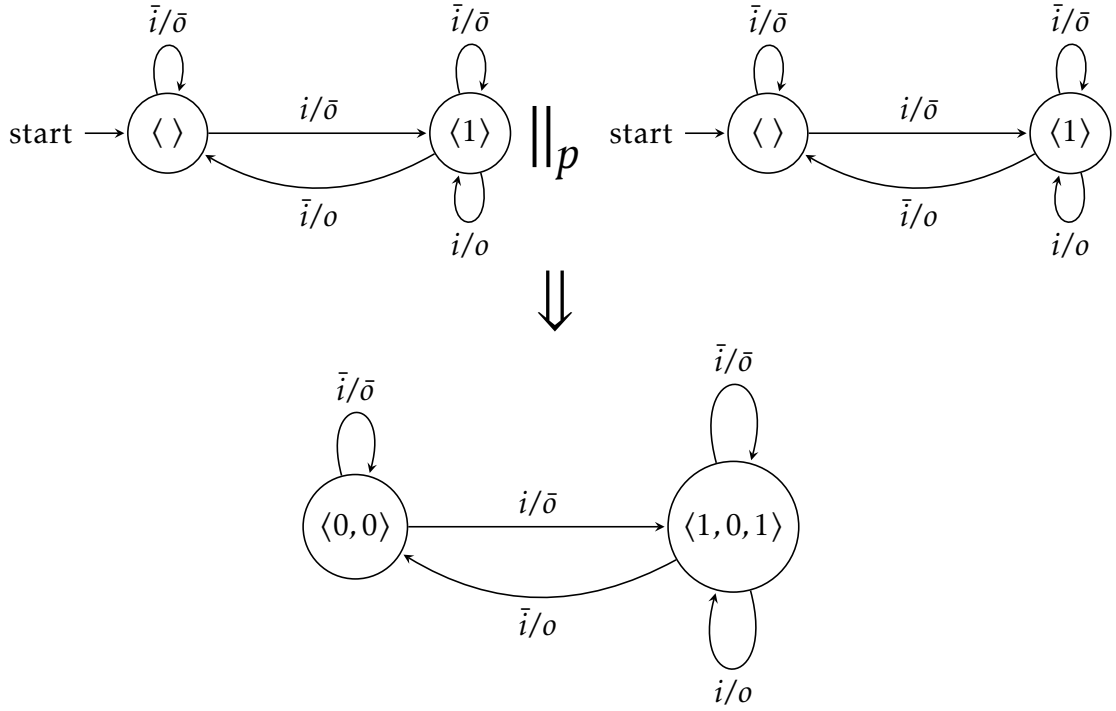

FIGURE 3.8 – Résultat de composition C (En bas) de deux SR-modèles $C_1(M_1 = 1, L_1 = 1)$\{Haut à gauche\} et C_2 $(M_2 = 1, L_2 = 1)$\{Haut à droite\}.

Description : L'automate global ci-dessus résulte de la composition parallèle de deux composants C_1 et C_2 ayant respectivement les paramètres $(M_1 = 1, L_1 = 1)$ et $(M_2 = 1, L_2 = 1)$.

Le composant résultat correspond au composant ayant les paramètres $M = 1, L = 1$, les états générés sont :

— L'état initial $\langle 0, 0 \rangle$: (les deux composants sont vides), dans lequel une boucle de silence $\bar{i}/\bar{o}$ est autorisée, ainsi qu'une transition $i/\bar{o}$ menant à l'état $\langle 1, 0, 1 \rangle$ qui correspond à la combinaison des deux états $\langle 1 \rangle$ et $\langle 1 \rangle$.

— L'état $\langle 1, 0, 1 \rangle$: correspond à l'état où les deux composants reçoivent une donnée. Trois transitions sont possibles à cet état :

— Une boucle $\bar{i}/\bar{o}$ qui signifie que les composants ont atteint leurs latences respectives et sont dans un état d'attente.

— Une boucle i/o car en produisant des sorties et en acceptant de nouvelles données en entrées il reste sur le même état.

— une transition $\bar{i}/o$ menant à $\langle 0, 0\rangle$, en produisant des sorties sans accepter d'entrées, les composants se vident et reviennent sur leurs états initiaux.

Remarque : Les états tels que $\langle 1, 0\rangle$ ou $\langle 0, 1\rangle$ ne sont pas générés car ils ne respectent pas les règles de synchronisation définies par rapport à l'étiquette d'entrée i. Le système ne compose que les transitions communes aux deux composants, ce qui garantit une cohérence parfaite en terme de mémoire et de latence, et génère ainsi un SR-modèle composite cohérent.

3.8 Conclusion

Ce chapitre a présenté notre approche de conception de systèmes réactifs synchrones corrects-par-construction. Nous avons introduit de nouvelles règles de composition parallèle pour les SR-modèles, qui permettent d'élaborer des systèmes complexes à partir de composants élémentaires tout en respectant les contraintes de mémoire et de latence. Notre contribution consiste en une définition formelle de la sémantique d'un SR-système, exprimée sous forme de règles de construction d'un système global à partir des SR-modèles de ses sous-composants. Cette définition intègre des contraintes garantissant la synchronisation appropriée des composants connectés en parallèle et recevant des données d'une source commune, comme dans une communication Multicast.

Le chapitre suivant analysera la validité de notre proposition à travers une implémentation en Python des règles de composition parallèle, ainsi qu'une étude de cas portant sur le véhicule autonome Cycab.

Chapitre 4

Evaluation de la composition parallèle - Use Case : Cycab

Introduction

Le chapitre précédent a présenté l'approche proposée reposant sur un nouvel opérateur de composition pour les SR-modèles. Nous y avons établi des contraintes de composition ainsi que de nouvelles règles permettant la génération de SR-modèles fiables tout en préservant la sémantique des SR-modèles.

Dans le chapitre actuel, nous allons évaluer notre approche sur une étude de cas : le véhicule autonome Cycab. Pour cela, dans un premier temps, nous allons présenter l'implémentation de notre approche en Python. Puis, nous allons modéliser les modules de Cycab. Enfin, nous allons effectuer une analyse pour valider notre approche.

4.1 Implémentation de la composition parallèle en Python

Dans le cadre de ce mémoire, nous nous sommes appuyés sur une implémentation des SR-modèles [14]. Cette implémentation automatise la génération des SR-modèles primitifs et la composition en série des SR-modèles. Nous avons enrichi cet outil en y intégrant la composition parallèle conforme à la définition 4. Cette extension permet de créer des SR-modèles par composition parallèle, fonctionnalité qui était absente de la version originale de l'outil.

4.1.1 Objectif de l'implémentation

L'outil développé permet l'automatisation de la modélisation des SR-modèles à deux niveaux :

— **Générer des SR-modèles primitifs** à partir de deux paramètres de mémoire M et de latence L.

— **Construire un SR-modèle composite :** à partir de deux SR-modèles primitifs caractérisés respectivement par (M_1, L_1) et (M_2, L_2). Le SR-modèle global est équivalent à un SR-modèle primitif caractérisé par (M, L).

L'automatisation de la génération des SR-modèles fournit une base pour la simulation, la vérification et la validation formelle de l'opérateur de composition proposé. Lorsqu'un SR-modèle est généré par composition parallèle $compose\_parallel(M_1, L_1, M_2, L_2)$, nous pouvons le comparer au SR-modèle primitif attendu $construct(M, L)$ afin de vérifier leur équivalence comportementale.

4.1.2 Bibliothèques utilisées

L'implémentation de notre outil de génération et de composition des SR-modèles s'appuie principalement sur la bibliothèque `Networkx`, une bibliothèque Python pour la manipulation de graphes orientés. Elle permet de représenter chaque automate sous forme de système de transitions étiquetées (Labelled Transition System — LTS), avec des nœuds pour les états et des arêtes pour les transitions.

```
import networkx as nx
from typing import List, Tuple, Set, Dict, Optional, Any, Callable
import copy
```

Figure 4.1 – Bibliothèques Python utilisées

— **networkx** : permet gestion des sommets, arêtes et opérations sur le graphe (successeurs, prédécesseurs, etc.)

— **typing** : permet de faire les annotations de types complexes (List, Tuple, Set...)

— **copy** : permet de cloner les listes de manière sécurisée.

4.1.3 Structure de données et classes utilisées

L'outil est organisé autour de trois classes principales : `LTS`, `LTSBuilder`, et `Composer`, chacune jouant un rôle spécifique dans la génération et la composition des SR-modèles.

— **LTS (Labelled Transition System)** : Cette classe représente un automate sous forme de graphe orienté. Elle encapsule les éléments suivants :

  — `LTSVertex` : qui représente un état du système, généralement encodé comme un tuple reflétant l'historique des entrées/sorties.

  — `LTSEdge` : représente une transition entre deux états, étiquetée par une paire entrée/sortie (exemple : i/o, $i/\bar{o}$,...).

  — Une interface de manipulation permettant d'ajouter des sommets et des transitions, d'exporter le graphe, et de le visualiser.

— **LTSBuilder** : Cette classe est responsable de la génération automatique d'un SR-modèle en utilisant la fonction *construct* à partir de deux paramètres (M, L) (Figure 4.2).Elle implémente un algorithme récursif qui :

  — Part d'un état initial vide comme nous pouvons le voir sur la figure 4.2 on initialise le premier état à *root* qui est la racine, on l'ajoute à la liste d'état à traiter, ainsi qu'à la liste d'états.

  — Génère toutes les transitions possibles selon l'algorithme primitif des SR-modèle qu'on n'a pas détaillé dans ce travail.

  Le résultat est un automate déterministe, fini, et conforme aux contraintes formelles des SR-modèles [14].

```python
class LTSBuilder:

    def __init__(self):
        self.lts = LTS()

    def empty(self) -> LTS:
        return LTS()

    def construct(self, root: LTSVertex, m: int, l: int) -> LTS:
        v1 = root
        unexplored = {root}
        states = {root}
```

FIGURE 4.2 – Début de la classe LTSBuilder + méthode construct()

— **Composer** : Cette classe englobe notre contribution, elle implémente l'opérateur de composition parallèle. Elle prend en entrée deux automates g_1 et g_2, et produit un automate composite g_C.

Elle repose sur la méthode `compose_parallel()`, qui initialise la composition en préparant les structures internes et en ajustant les paramètres mémoire et latence. Cette fonction appelle ensuite `compose_parallel_rules()`, laquelle parcourt les couples d'états accessibles pour appliquer les règles de synchronisation et de génération de transitions selon les contraintes des SR-modèles.

- La synchronisation s'effectue sur les actions communes.
- Les états du graphe composé sont des couples d'états des deux automates.
- Les transitions sont ajoutées en respectant les contraintes de compatibilité des actions.

Cette classe permet ainsi de construire un modèle global à partir de modules indépendants, tout en conservant la structure SR et le déterminisme.

4.1.4 Contribution : Composition parallèle

Nous allons voir un aperçu des éléments principaux de la fonction qui traduit la définition du nouvel opérateur de composition parallèle (définition 4).

a. Fonction `compose_parallel()`

Cette fonction initialise la composition parallèle en réinitialisant les structures internes, puis applique les règles via `LTSBuilder`. La Figure 4.3 illustre cette procédure sur deux automates avec leurs paramètres de mémoire et de latence respectifs.

```
def compose_parallel(self, g1: LTS, g2: LTS, m1: int, l1: int, m2: int, l2: int) -> LTS:
    self.gC.clear()
    self.stateC.clear()
    builder = LTSBuilder()
```

FIGURE 4.3 – Corps de la fonction compose-parallèle

b. `Contraintes de composition :`

Nous avons défini une condition d'exclusion qui permet de filtrer les combinaisons d'états qui violeraient les règles de cohérence entre automates. Elle prend en compte la synchronisation des comportements des deux SR-modèles en terme de mémoire et de latence. La Figure 4.4 illustre cette vérification en Python.

```
if (
    (find_maxI(src1.label) == 0 and find_maxI(src2.label) != 0) or
    (find_maxI(src1.label) != 0 and find_maxI(src2.label) == 0) or
    (len(dst1.label) != len(dst2.label)) or
    (find_maxI(src1.label) != find_maxI(src2.label) and (
        find_maxI(src1.label) < l1 or find_maxI(src2.label) < l2))
):
    return
```

FIGURE 4.4 – Condition d'exclusion

c. Mécanisme de génération de transitions

Lorsque la condition d'exclusion n'est pas satisfaite (figure **??**), la composition parallèle devient réalisable. Les deux exemples suivants illustrent ce processus de génération de transitions.

Génération de l'action $\bar{i}/\bar{o}$: Comme nous pouvons le voir sur la figure 4.5, cette condition vérifie que les deux composants sont dans un état de progression compatible avant de synchroniser deux transitions de type `("bar(i)", "bar(o)")`. Elle empêche les combinaisons déséquilibrées (par exemple, un composant actif et l'autre au repos, ou des longueurs de labels différentes).

```
if label1 == ("bar(i)", "bar(o)") and label2 == ("bar(i)", "bar(o)"):
    s1 = fresh(list(src1.label) + [0] + list(src2.label))
    s2 = fresh(list(dst1.label) + [0] + list(dst2.label))
    v1 = self._get_or_create_vertex(s1)
    v2 = self._get_or_create_vertex(s2)
    self.gC.add_edge(LTSEdge(v1, ("bar(i)", "bar(o)"), v2))
    self.stateC.add(v1)
    self.stateC.add(v2)
    return
```

FIGURE 4.5 – Génération de la transition $(\bar{i}, \bar{o})$ dans la fonction `_compose_parallel_rules`

Génération de l'action `i/o` : Ce cas correspond à une synchronisation entre les deux composants qui sont actifs su leurs interfaces d'entrées et de sorties (Figure 4.6). Lorsque chacun produit une transition (`"i"`, `"o"`), la composition est directe et acceptée sans aucune restriction.

```
if label1 == ("i", "o") and label2 == ("i", "o"):
    s1 = fresh(list(src1.label) + [0] + list(src2.label))
    s2 = fresh(list(dst1.label) + [0] + list(dst2.label))
    v1 = self._get_or_create_vertex(s1)
    v2 = self._get_or_create_vertex(s2)
    self.gC.add_edge(LTSEdge(v1, ("i", "o"), v2))
    self.stateC.add(v1)
    self.stateC.add(v2)
    return
```

FIGURE 4.6 – Génération de la transition (i, o) dans la fonction `_compose_parallel_rules`

4.2 Étude de cas : Application du modèle SR au système CyCab

Dans cette section, nous mettons en œuvre notre approche de modélisation fondée sur des SR-modèles (Systèmes Réactifs Synchrones) sur un système concret : le véhicule autonome CyCab, conçu par l'INRIA dans le cadre du projet IMARA [22]. Ce véhicule, qui vise à faciliter la mobilité urbaine automatisée, représente

un excellent exemple d'application pour évaluer notre outil de composition parallèle, en raison de sa structure modulaire et de ses exigences en temps réel.

4.2.1 Structure du système CyCab

Le CyCab est constitué de divers modules fonctionnels :

— **Le véhicule** : élément central, il transmet régulièrement des signaux de localisation et réagit selon les instructions reçues.

— **La station** : identifie la position du CyCab et lui transmet des ordres ($halt$! ou far!) en fonction de la distance.

— **Le bouton d'arrêt d'urgence** : peut à tout moment suspendre le fonctionnement du système.

— **Le starter** : Il est utilisé pour initier le départ du véhicule par un signal de commande.

Remarques sur les signaux utilisés :

— Le signal $position$! (émis par le véhicule) est une **sortie** pour ce composant, et devient une **entrée** $position$? pour la station.

— Les signaux $halt$! et far! sont des **sorties** de la station, et des **entrées** pour le véhicule.

— Le signal $emergency$! est une **sortie** du bouton d'urgence, généralement traité en entrée par le véhicule ou un module de supervision.

— Le signal $start$! (starter) est une commande initiale, probablement une sortie vers le véhicule.

— Le signal $reset$! Ce signal est souvent utilisé pour réinitialiser le système, donc on le considère généralement comme une sortie d'un composant de contrôle, et entrée pour le véhicule.

Ces interactions, représentées dans la figure 4.7, sont fondées sur un modèle événementiel synchrone. Elles justifient le recours à une modélisation réactive à mémoire et latence (SR-modèles), car chaque composant peut être décrit de manière autonome à l'aide d'automates i/o synchronisés dans un cadre global.

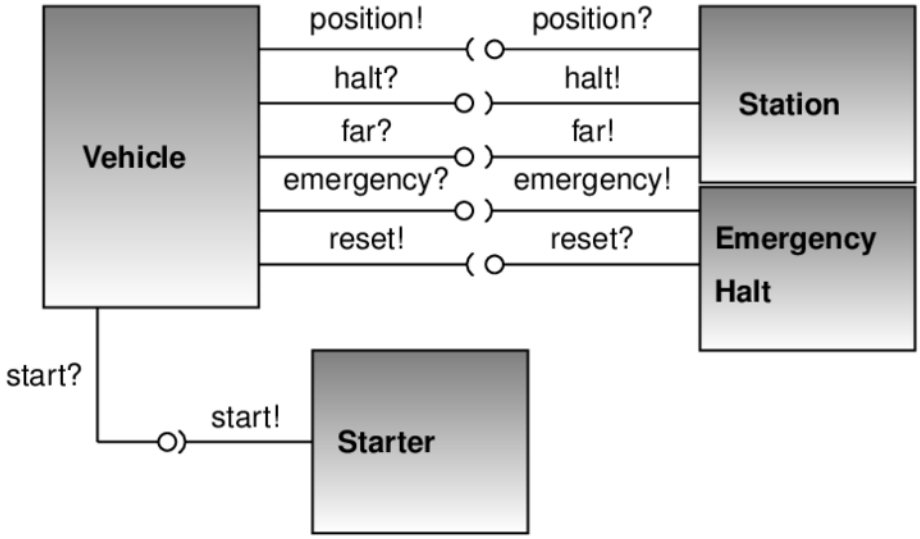

FIGURE 4.7 – Le modèle UML des composants de Cycab [15]

4.2.2 Architecture modulaire du Cycab

Le comportement de chaque composant fonctionnel du Cycab peut être formellement spécifié à l'aide des paramètres de mémoire M et de latence L, caractéristiques des SR-modèles. Nous modélisons ainsi chaque module du système de manière indépendante, en tenant compte des signaux qu'il reçoit ou émet :

— **Bouton d'arrêt d'urgence** : ce composant produit un signal *emergency*! à tout moment. Il peut également recevoir un signal *reset*? pour réinitialiser son état. Il ne dépend que de l'état courant et ne nécessite pas de mémoire historique complexe.

⇒ Modélisé avec $(M = 1, L = 1)$

— **Station** : elle reçoit le signal *position*? provenant du véhicule et envoie en réponse un signal *halt*!, *far*! ou *reset*!. Ce module effectue une action en fonction de l'état du véhicule à un instant donné, sans mémoire d'historique prolongée.

⇒ Modélisé avec $(M = 1, L = 1)$

— **Véhicule** : il reçoit trois types de signaux : *halt*?, *far*? et *reset*?. Il adapte ses actions en fonction de ces entrées, en conservant une trace des commandes reçues. Ainsi, il a besoin de stocker jusqu'à trois valeurs d'entrée.

$\Rightarrow$ Modélisé avec $(M = 3, L = 1)$

Ces composants sont modélisés dans notre outil Python comme des **modèles réactifs synchrones (SR-modèles)**, et sont représentés sous forme de graphes orientés à l'aide de la classe `LTSBuilder`.

4.3 Modélisation de Cycab par les SR-modèles

Pour évaluer la pertinence de notre méthode de composition parallèle fondée sur les SR-modèles, nous avons appliqué notre implémentation Python au cas d'étude du véhicule autonome Cycab, développé dans le cadre du projet IMARA – INRIA [22].

Cette modélisation repose sur deux étapes principales :

— **Modélisation individuelle des composants** : chaque module fonctionnel (station, véhicule, bouton d'arrêt d'urgence, etc.) est représenté séparément sous forme d'un SR-modèle, caractérisé par une paire (M_i, L_i).

— **Composition des composants** : les automates sont ensuite combinés pour générer un modèle global du système.

Les paramètres globaux (M, L) dépendent du type de connexion entre les composants :

— *Dans le cas d'une composition parallèle*, les calculs se basent sur les conditions formelles établies dans le chapitre 3, section *Conditions de composition parallèle*, à savoir :

$$M = \min(M_1, M_2) \quad \text{et} \quad L = \begin{cases} \max(L_1, L_2) & \text{si } L_1 > 1 \text{ ou } L_2 > 1 \\ 1 & \text{si } L_1 = 1 \text{ ou } L_2 = 1 \end{cases}$$

Ces conditions assurent que le système global reste réactif, même si un seul composant possède une faible latence, tout en maintenant la contrainte mémoire la plus stricte.

— *Dans le cas d'une composition en série*, les composants sont reliés de manière séquentielle (pipeline). Dans ce cas, les paramètres globaux sont obtenus par

sommation :

$$M = \sum M_i, \quad L = \sum L_i$$

Cette règle traduit l'accumulation de mémoire et de délais entre chaque étape du système.

Cette distinction permet une modélisation plus fidèle à l'architecture réelle du système, en tenant compte des contraintes spécifiques à chaque type d'interconnexion.

4.3.1 Modélisation des SR-modèles primitifs

Dans un premier temps, nous avons modélisé séparément chacun des composants du système Cycab sous forme de SR-modèles primitifs, en spécifiant pour chacun une paire de paramètres (M, L) adaptée à son comportement réactif.

Cette modélisation a été effectuée en utilisant notre outil Python, via le module `lts-builder`, qui génère automatiquement les automates associés.

— **Bouton d'arrêt d'urgence** : Ce composant réagit immédiatement à une activation sans historique.Le SR-modèle correspondant est sur la figure 4.8. Il reçoit des signaux tels que `reset!`, ce qui justifie une modélisation avec :

$$M = 1, \quad L = 1$$

```
Choix :1
Donnez la valeur de la mémoire du composant :1
Donnez la valeur de la latence du  composant :1
Creating component ...
Component 1:
LTS with 2 vertices and 5 edges:
vertex 1
edges 3
  edge 1 --('i', 'o')--> 1
  edge 1 --('bar(i)', 'o')-->
  edge 1 --('bar(i)', 'bar(o)')--> 1
vertex
edges 2
  edge  --('i', 'bar(o)')--> 1
  edge  --('bar(i)', 'bar(o)')-->
End Graph
```

Figure 4.8 – SR-modèle de bouton d'arrêt d'urgence

— **Station** : Elle reçoit des signaux de position `position?` du véhicule et répond par `halt!` ou `far!`. La figure 4.14 montre son SR-modèle ,le traitement se fait sans accumulation de signaux, ce qui correspond aux paramètres :

$$M = 1, \quad L = 1$$

```
LTS with 2 vertices and 5 edges:
vertex 1
edges 3
  edge 1 --('i', 'o')--> 1
  edge 1 --('bar(i)', 'o')-->
  edge 1 --('bar(i)', 'bar(o)')--> 1
vertex
edges 2
  edge  --('i', 'bar(o)')--> 1
  edge  --('bar(i)', 'bar(o)')-->
End Graph
```

Figure 4.9 – SR-modèle de la station

— **Véhicule** : Ce composant est le plus complexe. Il reçoit plusieurs signaux (`halt?`, `far?`, `reset?`), ce qui nécessite une capacité de mémoire supérieure pour mémoriser les différentes instructions. Nous montrons dans la figure 4.10 uniquement une partie de l'automate. Il est modélisé avec :

$$M = 3, \quad L = 1$$

```
LTS with 4 vertices and 13 edges:
vertex 1
edges 4
  edge 1 --('i', 'o')--> 1
  edge 1 --('i', 'bar(o)')--> 1;1
  edge 1 --('bar(i)', 'o')-->
  edge 1 --('bar(i)', 'bar(o)')--> 1
vertex 1;1
edges 4
  edge 1;1 --('i', 'o')--> 1;1
  edge 1;1 --('i', 'bar(o)')--> 1;1;1
  edge 1;1 --('bar(i)', 'o')--> 1
  edge 1;1 --('bar(i)', 'bar(o)')--> 1;1
vertex
```

Figure 4.10 – SR-modèle du véhicule

Chacun de ces automates a été construit à l'aide de l'interface textuelle de notre outil Python, puis visualisé avec la fonction `print_graph()` pour vérification et analyse.

4.3.2 Construction du SR-modèle de Cycab par Composition

Nous avons appliqué notre approche de composition parallèle pour obtenir un SR-modèle global représentant le comportement composite :

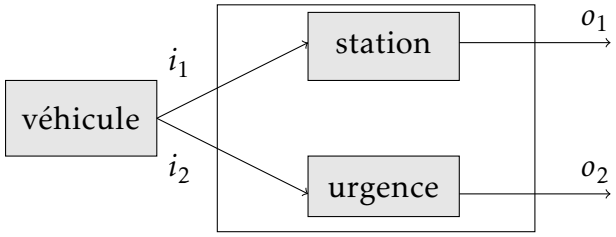

FIGURE 4.11 – Connexion en parallèle type Multicast(Station + Urgence)

1. **Composition du bouton d'arrêt et de la station (parallèle)** :

   — **Composants** : $(M = 1, L = 1)$ et $(M = 1, L = 1)$

   — **Résultat** : $M = \min(1,1) = 1$, $L = 1$

```
LTS with 2 vertices and 5 edges:
vertex 0
edges 2
  edge 0 --('i', 'bar(o)')--> 1;0;1
  edge 0 --('bar(i)', 'bar(o)')--> 0
vertex 1;0;1
edges 3
  edge 1;0;1 --('i', 'o')--> 1;0;1
  edge 1;0;1 --('bar(i)', 'o')--> 0
  edge 1;0;1 --('bar(i)', 'bar(o)')--> 1;0;1
End Graph
```

FIGURE 4.12 – SR-modèle obtenu par composition parallèle (bouton d'arrêt + la station)

2. **Composition du véhicule avec la (station + bouton d'arrêt) (série)** :

   — **Composants initiaux** : $(M = 2, L = 1)$ et $(M = 1, L = 1)$

   — **Composition résultante** : $M = 2 + 1 = 3$, $L = 1 + 1 = 2$

— **Interprétation** : Le système résultant de la composition **série** est modélisé par un système de transitions étiquetées (LTS) comportant 5 états et 14 transitions.

```
LTS with 5 vertices and 14 edges:
vertex 0;1
edges 4
  edge 0;1 --('i', 'o')--> 1;0
  edge 0;1 --('i', 'bar(o)')--> 1;0;1
  edge 0;1 --('bar(i)', 'o')--> 0
  edge 0;1 --('bar(i)', 'bar(o)')--> 0;1
vertex 1;0;1
edges 3
  edge 1;0;1 --('i', 'o')--> 1;0;1
  edge 1;0;1 --('bar(i)', 'o')--> 0;1
  edge 1;0;1 --('bar(i)', 'bar(o)')--> 1;0;1
vertex 1;0
```

FIGURE 4.13 – SR-modèle obtenu par composition en série (véhicule + (la station+bouton d'arrêt))

3. **Composition du Starter et du Véhicule (série)** :

— **Composants** : $(M = 1, L = 1)$ (Starter) et $(M = 3, L = 1)$ (Véhicule)

— **Résultat** : $M = M_1 + M_2 = 1 + 3 = 4, \quad L = L_1 + L_2 = 1 + 1 = 2$

— **Interprétation** : Le système obtenu par *composition série* de ces deux composants est modélisé par un automate à états finis (LTS) comportant **6 états** et **17 transitions**.

```
Serial Composing components ...
-------Component composite-------
LTS with 6 vertices and 17 edges:
vertex 0;1
edges 4
  edge 0;1 --('i', 'o')--> 1;0
  edge 0;1 --('i', 'bar(o)')--> 1;0;1
  edge 0;1 --('bar(i)', 'o')--> 0
  edge 0;1 --('bar(i)', 'bar(o)')--> 0;1
vertex 1;0;1
edges 3
  edge 1;0;1 --('i', 'o')--> 1;0;1
  edge 1;0;1 --('bar(i)', 'o')--> 0;1
  edge 1;0;1 --('bar(i)', 'bar(o)')--> 1;0;1
```

FIGURE 4.14 – SR-modèle obtenu par composition en série(Starter + du Véhicule)

4. **Composition finale (SR-modèle de Cycab par composition) : (Starter+véhicule) composé en série avec (bouton d'arrêt+station)** :
   - **Composants** :
     - Composition série de Starter ($M = 1, L = 1$) et Véhicule ($M = 3, L = 1$) $\Rightarrow$ ($M = 4, L = 2$)
     - Composition parallèle de Bouton d'arrêt ($M = 1, L = 1$) et Station ($M = 1, L = 1$) $\Rightarrow$ ($M = 1, L = 1$)
   - **Résultat** : $M = M_1 + M_2 = 4 + 1 = 5, \quad L = L_1 + L_2 = 2 + 1 = 3$
   - **Interprétation** : Le SR-modèle global, obtenu par composition série + parallèle, décrit le comportement du système Cycab complet avec les paramètres $M = 5$ et $L = 3$. L'automate correspondant comprend **10 états** et **24 transitions**.

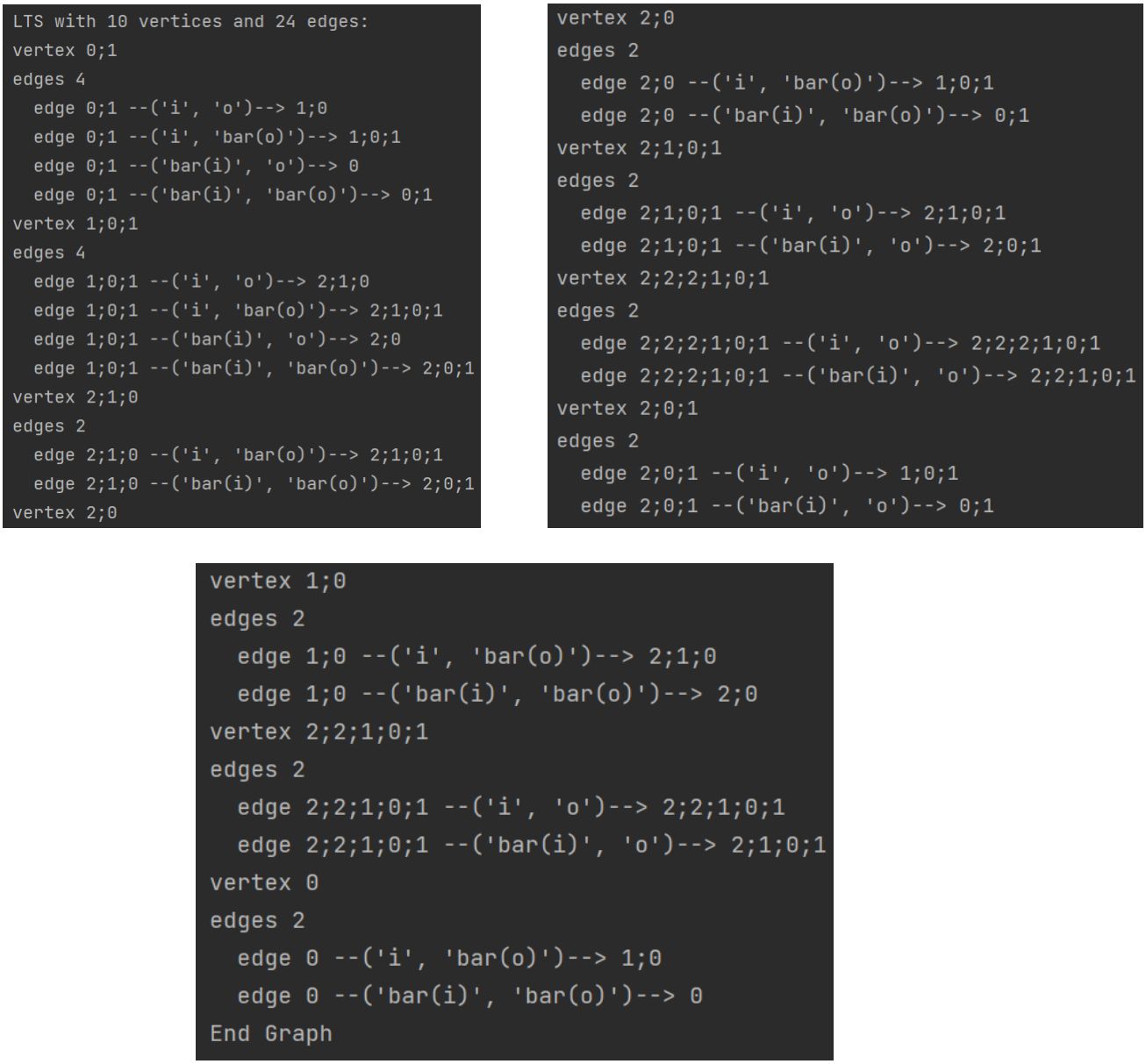

Figure 4.15 – SR-modèle obtenu par Composition de Cycab

4.3.3 Analyse de la composition parallèle des SR-modèles

Afin de valider l'approche proposée dans ce travail, il est essentiel de comparer les modèles obtenus par composition parallèle avec les modèles primitifs attendus. Cette comparaison permet de vérifier si la composition préserve bien les propriétés fondamentales des SR-modèles, notamment celles liées à la mémoire (M), à la latence (L), et au déterminisme.

Le tableau ci-dessous 4.1 présente une synthèse comparative entre deux approches :

- D'une part, le **modèle composé**, obtenu en combinant en parallèle deux SR-modèles

élémentaires : le bouton d'urgence et la station.

- D'autre part, le **modèle primitif** attendu, généré directement avec les paramètres attendu $M = 1$ et $L = 1$.

Critère	Modèle composite (Parallel Composition)	Modèle Primitif(SR-Model attendu)
Nom du composant	Bouton d'urgence $\|\|_p$ Station	Modèle SR global avec $M = 1, L = 1$
Nombre d'états	2 états	2 états
Nombre de transitions	5 transitions	5 transitions
Correspondance des états	<0,0> <1,0,1>	< > <1>
Déterminisme	Oui	Oui

TABLE 4.1 – Comparaison entre le modèle composé (bouton d'urgence + station en parallèle) et le SR- modèle attendu avec $M = 1$, $L = 1$

4.4 Analyse de la combinaison de notre contribution et la composition en série

Nous avons d'une part analysé la partie qui concerne notre proposition :la composition parallèle. Le reste de la composition des composants de Cycab se fait par série. Nous montrons dans le tableau 4.2 que notre opérateur de composition parallèle se combine sans problèmes à la composition en série des SR-modèles. Le tableau montre que le SR-modèle attendu de CyCab correspond au SR-modèle généré par composition de tous ses sous-composants

Critère	Composition	SR-Modèle attendu
Nom du composant	Starter || Véhicule || (Station $\|\|_p$ Bouton d'urgence)	SR-Modèle global de Cycab $M = 5$ $L = 3$
Nombre d'états	10	10
Nombre de transitions	24	24
Déterminisme	Oui	Oui

TABLE 4.2 – Comparaison entre le SR-modèle composite de Cycab et le SR-modèle attendu de Cycab avec $M = 5$, $L = 3$

4.5 Discussion des résultats

Les résultats obtenus démontrent que la composition parallèle proposée produit effectivement le comportement attendu. En ajoutant la composition en parallèle et en la combinant avec la composition en série, notre approche permet de représenter intégralement le composant Cycab dans un modèle unifié. Cette capacité de modélisation globale constitue un avantage significatif par rapport aux travaux de [15], qui, en

se limitant uniquement à la composition série, n'ont pu modéliser le composant que de manière fragmentée, contraignant ainsi l'analyse à une approche partie par partie. L'introduction de la composition parallèle enrichit donc considérablement les possibilités de modélisation en permettant une représentation complète et cohérente des systèmes synchrones réactifs complexes.

4.6 Conclusion

Dans ce chapitre, nous avons d'une part implémenté l'approche proposée dans un outil de génération automatique de SR-modèles, et d'autre part, nous avons vérifié notre approche sur une étude de cas réelle : véhicule autonome Cycab.

L'étude de cas du Cycab a d'abord nécessité une modélisation individuelle de chaque composant fonctionnel du système par les SR-modèles. Cette phase nous a permis de démontrer l'efficacité de notre approche de composition parallèle pour la modélisation d'un sous-ensemble du système (Station et bouton d'urgence). En combinant cette technique avec la composition en série, nous sommes parvenus à élaborer le SR-modèle global de Cycab, montrant ainsi la complémentarité de notre modélisation avec la modélisation compositionnelle existante des SR-modèles.

Cette analyse représente une première étape vers la validation de notre approche de composition en parallèle des SR-modèles. Bien que limitée à une architecture de système spécifique, elle démontre la capacité de notre approche compositionnelle à reproduire fidèlement le comportement attendu.

Conclusion générale

Ce mémoire a exploré une approche formelle et modulaire pour la modélisation de systèmes réactifs, en s'appuyant sur des automates I/O enrichis de deux paramètres fondamentaux : la mémoire (M) et la latence (L), appelés SR-modèles. Ce formalisme convient aux applications où seul le transfert de données importe, indépendamment de leur valeur réelle, comme le traitement de flux de données ou de signal.
Après avoir passé en revue les bases théoriques et analysé de manière critique les modèles existants, notre choix s'est orienté vers les SR-modèles, particulièrement adaptés aux systèmes réactifs que nous étudions.

Nous avons étendu la théorie des SR-modèles avec un nouvel opérateur de composition parallèle, qui permet d'assembler dynamiquement des composants élémentaires tout en garantissant des propriétés essentielles des SR-systèmes telles que le *déterminisme* et la *réutilisabilité*, et la *correction-par-construction*.

Nous avons implémenté notre approche par un outil en Python, avec lequel nous avons fait l'expérimentation sur un système réel — le véhicule autonome Cycab — Cette expérimentation a permis de vérifier la correction de notre approche. Nous avons montré qu'il était possible de reconstituer un comportement global cohérent à partir de composants primitifs en utilisant notre opérateur de composition.

Cette approche s'appuie sur une stratégie de conception ascendante (*bottom-up*), dans laquelle l'analyse et la validation de modules indépendants précèdent leur assemblage, garantissant ainsi la préservation de la correction du système global.

Perspectives

Ce mémoire ouvre plusieurs pistes de prolongement et d'améliorations du travail effectué. La conception par composition parallèle constitue une classe importante des systèmes réactifs. Cependant, la proposition peut être étendue afin de prendre en compte les points suivants :

— **Validation formelle** : L'étape suivante serait de vérifier l'opérateur de composition proposé avec des outils de Model Checking, et de Theorem proving afin de valider formellement que la proposition repsectent les propriétés comportementales des SR-modèles.

— **Généralisation à d'autres types de composition** : intégrer la *hiérarchique*, afin de couvrir un éventail plus large de scénarios d'architecture logicielle.

— **Extension vers des événements concurrentiels** : introduire des mécanismes de *gestion de conflits* ou de *synchronisation partielle*, notamment dans les systèmes où plusieurs entrées peuvent survenir simultanément.

— **Application à d'autres systèmes critiques** : tester notre approche sur d'autres types de systèmes (IOT, Pipelines de machine learning ...)

Bibliographie

[1] A. Arnold. *Finite Transition Systems : Semantics of Communicating Systems*. Prentice-Hall, 1994.

[2] Paul C. Attie. "Dynamic Input/Output Automata : A Formal Model for Dynamic Systems". In : *Proceedings of the International Conference on Concurrency Theory (CONCUR)*. Springer, 1999, p. 292-307.

[3] Algirdas Avizienis et al. "Basic Concepts and Taxonomy of Dependable and Secure Computing". In : *IEEE Transactions on Dependable and Secure Computing* 1.1 (2004), p. 11-33.

[4] Radhakisan Baheti et Helen Gill. "Cyber-Physical Systems". In : *The Impact of Control Technology* 12.1 (2011), p. 161-166. url : https://ieeecss.org/sites/ieeecss/files/2019-07/IoCT-Part3-02CyberphysicalSystems.pdf.

[5] Julien Bastian, Olivier H. Roux et Joseph Sifakis. "Modular Construction of Petri Net Models for Real-Time Systems". In : *Journal of Systems Architecture* 47.9 (2001), p. 729-748.

[6] A. Basu, M. Bozga et J. Sifakis. "Modeling Heterogeneous Real-time Components in BIP". In : *Proceedings of the 4th IEEE International Conference on Software Engineering and Formal Methods (SEFM)*. IEEE, 2006, p. 3-12.

[7] Gérard Berry. "Esterel on the Way to Industrialization". In : *Proceedings of the IEEE*. T. 79. 9. IEEE, 1991, p. 1293-1306.

[8] Gérard Berry et Georges Gonthier. "The Synchronous Programming Language ESTEREL : Design, Semantics, Implementation". In : *Science of Computer Programming* 19.2 (1992), p. 87-152.

[9] Bernard Berthomieu et François Vernadat. "Time Petri Nets Analysis with Tina". In : *Proceedings of the 3rd International Conference on the Quantitative Evaluation of Systems (QEST)*. IEEE, 2006, p. 123-124.

[10] Bernard Berthomieu et al. "Formal Methods in Safety-Critical Systems Engineering". In : *FM 2008 : Formal Methods*. Springer, 2008, p. 192-205.

[11] Brewminate. *The Internet at 50 : The Night the Internet Was Born*. n.d. URL : https://brewminate.com/the-internet-at-50-the-night-the-internet-was-born/.

[12] S. Chabane. "Composants virtuels : abstraction, vérification et réutilisation". Thèse de doct. Université M'hamed Bougara - Boumerdes, 2021.

[13] Sarah Chabane, Rabéa Ameur-Boulifa et Mohamed Mezghiche. "Rethinking of I/O-automata composition". In : *Forum on specification and Design Languages (FDL)*. IEEE, 2017, p. 1-7.

[14] Sarah Chabane, Rabéa Ameur-Boulifa et Mezghiche Mohamed. "Towards compositional verification of synchronous reactive systems". In : *International Journal of Critical Computer-Based Systems* 10.2 (2021), p. 120-142.

[15] Samir Chouali, Hassan Mountassir et Sebti Mouelhi. "An I/O automata-based approach to verify component compatibility : application to the CyCab car". In : *Electronic Notes in Theoretical Computer Science* 238.6 (2010), p. 3-13.

[16] Johan Eker et al. "Taming Heterogeneity—the Ptolemy Approach". In : *Proceedings of the IEEE*. T. 91. 1. IEEE, 2003, p. 127-144.

[17] N. Halbwachs et al. "The synchronous data flow programming language Lustre". In : *Proceedings of the IEEE* 79.9 (1991), p. 1305-1320.

[18] Nicolas Halbwachs. *Synchronous Programming of Reactive Systems*. Boston : Kluwer Academic Publishers, 1993.

[19] Nicolas Halbwachs et al. "Lustre : A Declarative Language for Programming Synchronous Systems". In : *Proceedings of the 14th ACM SIGACT-SIGPLAN symposium on Principles of programming languages* (1987), p. 178-188.

[20] David Harel et Amir Pnueli. "On the Development of Reactive Systems". In : *Logics and Models of Concurrent Systems*. Springer, 1985, p. 477-498. DOI : 10.1007/978-1-4612-1024-8_22.

[21] Industry USA Magazine. *Figure 2 : Major IVN Technologies*. https://cdn.induportals-media-publishing.org/Press+Files/02/81/41/28141-ONSAR3014-Figure-2-Major-IVN-Technologies.jpg.

[22] INRIA. *IMARA Project – Intelligent Mobility and Automotive Research*. https://team.inria.fr/imara/. Consulté en 2025. 2020.

[23] ITU-T. *Recommendation Z.100 : Specification and Description Language (SDL)*. https://www.itu.int/rec/T-REC-Z.100. International Telecommunication Union. 2016.

[24] G. Kahn. "The Semantics of a Simple Language for Parallel Programming". In : *Proceedings of the IFIP Congress*. 1974.

[25] Hamoudi Kalla. *Modèle d'un système réactif*. Figure 1 dans l'article "Génération automatique de distributions/ordonnancements temps réel, fiables et tolérants

aux fautes". 2004. URL : https://www.researchgate.net/figure/Modele-dun-systeme-reactif_fig1_30512398.

[26] Dilsun K Kaynar et al. "Timed I/O automata : a formal foundation for modeling and analysis of real-time systems". In : *Real-Time Systems* 33.1-3 (2006), p. 73-127.

[27] KPN. *Logo officiel de KPN*. 2018. URL : https://spokenagency.nl/wp-content/uploads/2018/11/KPN-logo-vierkant.jpg.

[28] Paul Le Guernic et al. "Signal : A Data Flow-Oriented Language for Signal Processing". In : *IEEE Transactions on Acoustics, Speech, and Signal Processing* 34.2 (1991), p. 362-374.

[29] Edward A. Lee. "Cyber Physical Systems : Design Challenges". In : *11th IEEE International Symposium on Object Oriented Real-Time Distributed Computing (ISORC)*. IEEE, 2008, p. 363-369.

[30] Edward A. Lee et David G. Messerschmitt. "Synchronous data flow". In : *Proceedings of the IEEE*. T. 75. 9. IEEE, 1987, p. 1235-1245.

[31] John Lygeros, Shankar Sastry et Claire J Tomlin. "Dynamical properties of hybrid automata". In : *Hybrid Systems : Computation and Control*. Springer. 2003, p. 7-23.

[32] N. A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.

[33] Nancy Lynch et Roberto Segala. "Probabilistic input/output automata". In : *Information and Computation* 168.2 (2001), p. 108-160.

[34] Nancy Lynch, Roberto Segala et Frits Vaandrager. "Hybrid I/O Automata Revisited". In : *Hybrid Systems : Computation and Control* 2623 (2003), p. 403-417.

[35] Nancy Lynch et Mark Tuttle. *An introduction to input/output automata*. MIT Press, 1989.

[36] Nancy A. Lynch et Mark R. Tuttle. "Hierarchical correctness proofs for distributed algorithms". In : *Proceedings of the 6th ACM Symposium on Principles of Distributed Computing (PODC)*. 1987, p. 137-151.

[37] Xavier Maranget. *Introduction aux automates*. https://gallium.inria.fr/~maranget/X/421/poly/automate.html. Consulté en 2025. 2024.

[38] Alain J. Martin. "The Design of an Asynchronous Microprocessor". In : *Advanced Research in VLSI : Proceedings of the Decennial Caltech Conference on VLSI*. Sous la dir. de C. Seitz. Cambridge, MA : MIT Press, 1989, p. 351-373.

[39] Microsoft Learn. *Asynchronous Event-Driven Communication in Microservices*. https://learn.microsoft.com/en-us/dotnet/architecture/microservices/architect-microservice-container-applications/asynchronous-message-based-communication. 2025.

[40] Carl Adam Petri. “Kommunikation mit Automaten”. Thèse de doctorat. Thèse de doct. Technische Hochschule Darmstadt, 1962.

[41] Shaz Qadeer, Sriram K Rajamani et Sanjit A Seshia. “Dynamic input/output automata : A model for dynamic systems”. In : *arXiv preprint arXiv :2003.13437* (2020).

[42] Roberto Segala. “Modeling and Verification of Randomized Distributed Real-Time Systems”. PhD thesis. Massachusetts Institute of Technology (MIT), 1995.

[43] Joseph Sifakis. “A Framework for Component-based Construction”. In : *Construction and Analysis of Safe, Secure, and Interoperable Smart Devices* 3364 (2005), p. 293-312.

[44] SlidePlayer. *Modes de transmission : Synchrone - Asynchrone*. https://images.slideplayer.fr/118/18158175/slides/slide_6.jpg. 2019.

[45] Frits W. Vaandrager et Nancy A. Lynch. “Timed I/O Automata : A Formalism for the Specification and Verification of Real-Time Systems”. In : *NATO ASI Series F : Computer and Systems Sciences* 138 (1995), p. 387-423.

[46] Luigi Zaffalon. *SCADE - Modélisation formelle de systèmes réactifs critiques*. 2018. url : https://hal.archives-ouvertes.fr/hal-01835599v2.